

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/351777065>

Desarrollo web

Book · May 2021

CITATIONS

0

READS

1,340

1 author:



[Jorge Domínguez Chávez](#)

Universidad Politécnica Territorial del Estado Aragua

68 PUBLICATIONS 42 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Curso de Programación Android con Java [View project](#)



informática [View project](#)

DESARROLLO DE APLICACIONES WEB

JORGE DOMÍNGUEZ CHÁVEZ

Copyright © Jorge Domínguez Chávez.

ORCID: 0000-0002-5018-3242

Esta obra se distribuye licencia Creative Commons:



<http://creativecommonsvenezuela.org.ve>

Reconocimiento:

- Atribución: Permite a otros copiar, distribuir, exhibir y realizar su trabajo con Derechos de Autor y trabajos derivados basados en ella – pero sólo si ellos dan créditos de la manera en que usted lo solicite.
- Compartir igual: Permite que otros distribuyan trabajos derivados sólo bajo una licencia idéntica a la que rige el trabajo original.
- Adaptar: mezclar, transformar y crear a partir de este material.

Siempre que lo haga bajo las condiciones siguientes:

- Reconocimiento: reconocer plenamente la autoría de Jorge Domínguez Chávez, proporcionar un enlace a la licencia e indicar si se ha realizado cambios. Puede hacerlo de cualquier manera razonable, pero no una que sugiera que tiene el apoyo del autor o lo recibe por el uso que hace.
- No Comercial: no puede utilizar el material para una finalidad comercial o lucrativa.

© 2019, Domínguez Chávez, Jorge

ISBN 69-78-9806-366008



Publicado por IEASS, Editores

ieass@blogspot.com

Venezuela, 2019

La portada y contenido de este libro fue editado y maquetado en TeXstudio 2.12.10

Indice

1. Servidores de Aplicaciones WEB	1
1.1. Servidores	1
1.2. Servidores web	2
1.3. Servidores de aplicaciones web	2
1.4. Tecnologías para crear aplicaciones web	3
1.4.1. Apache	3
1.4.2. NGINX	4
1.4.3. PHP (Personal Home Pages)	4
1.4.4. Contraseñas de usuarios	4
1.4.5. Mariadb	5
1.4.6. html5	5
1.4.7. ¿Cómo funciona el XHTML?	6
1.4.8. Css3	6
1.4.9. JavaScript	7
1.4.10. Bootstrap	8
2. Primeros ejemplos	11
2.1. html5	11
2.1.1. Editor de texto	11
2.1.2. Ruta o Árbol de navegación	11
2.2. Programa Hola, mundo	12
2.3. Funcionamiento básico de CSS	12
2.3.1. Incluir CSS en el mismo documento HTML	13
2.3.2. Definir CSS en un archivo externo	14
2.3.3. Incluir CSS en los elementos HTML	16
2.4. Javascript	16
2.4.1. JavaScript en documentos HTML	16
2.5. De nuevo HTML	17
2.6. Archivos con código JavaScript	17
2.7. Sin sentencia HTML	18
2.8. Bootstrap	18
2.8.1. Sistema de cuadrilla y diseño sensible	18
2.8.2. Entendiendo la hoja de estilo CSS	18
2.8.3. Creando una cuadrilla de diseño fija	19
2.8.4. Creando una cuadrilla de diseño fija con una cuadrilla de diseño fluida anidada	19

3. La programación	20
3.1. Qué es el HTML?	20
3.2. Y eso de CSS?	20
3.3. Características del lenguaje HTML	20
3.4. Estructura HTML	21
3.4.1. La codificación	21
3.4.2. La plantilla	22
3.4.3. Etiquetas básicas	22
3.4.4. Tablas para layouts?	31
3.5. CSS	31
3.5.1. Dónde escribo el código CSS?	31
3.5.2. Trucos con CSS	38
3.5.3. Campos de formulario	40
3.5.4. heading	43
3.5.5. Enlace	43
3.5.6. Listas personalizadas	44
3.5.7. Lista	44
3.5.8. Los ítem	44
3.5.9. Menús verticales	45
3.6. Entorno	46
3.7. Migración a HTML5	47
3.7.1. Minúsculas y comillas	47
3.7.2. Cierre todas las etiquetas	48
3.7.3. Hay que usar alt y title	48
3.7.4. Cuidado al anidar etiquetas	48
3.7.5. No existen los frames	49
3.8. instancias del navegador	49
3.8.1. Las tablas no se usan para maquetar	49
3.8.2. Los ampersands	49
3.8.3. Utiliza Unicode	50
3.9. Formularios	50
3.9.1. La etiqueta FORM	50
3.9.2. Campos de texto	51
3.9.3. Campos de contraseña	51
3.9.4. Etiquetar campos	51
3.9.5. Áreas de texto	52
3.9.6. Casillas de verificación	52
3.9.7. Botones de selección	53
3.9.8. Listas de selección	54
3.9.9. Botones de enviar y reestablecer	54
3.10. Ejercicios	55
3.10.1. Práctica 1	55
3.10.2. Práctica 2	55

4. Diseño de Base de datos	57
4.1. Objetivos	57
4.2. Qué es una base de datos?	57
4.3. Etapas del diseño de bases de datos	57
4.4. Diseño conceptual: modelo Entidad-Relación	58
4.4.1. Entidades, atributos e relaciones	58
4.4.2. Grado de las relaciones	61
4.4.3. Caso de estudio: base de datos de casas de colonias	63
4.4.4. Aspectos del modelo ER	65
4.5. Ejercicios	66
5. Base de datos, SQL y MariaDB	69
5.1. Entrando y saliendo de MariaDB	69
5.2. Creación de una base de datos	69
5.3. Consultas básicas con una tabla	70
5.3.1. Creación de una tabla	70
5.3.2. Introducción de datos	70
5.3.3. Mostrar datos	71
5.3.4. Buscar por contenido	71
5.3.5. Mostrar datos ordenados	72
5.3.6. Salir de MySQL	72
5.3.7. Ejercicios propuestos	72
5.4. Consultas básicas con dos tablas	73
5.4.1. Formalizando conceptos	73
5.4.2. Por qué varias tablas?	74
5.4.3. Las claves primarias	74
5.4.4. Creando datos	75
5.4.5. Mostrando datos	76
5.4.6. Ejecutando un lote de órdenes	77
5.4.7. Ejercicios propuestos	78
5.5. Borrar información	78
5.5.1. Qué información hay?	78
5.5.2. Borrar toda la base de datos	79
5.5.3. Borrar una tabla	79
5.5.4. Borrar datos de una tabla	79
5.5.5. Ejercicios propuestos	79
5.6. Modificar información	80
5.6.1. Modificación de datos	80
5.6.2. Ejercicios propuestos sobre modificación de datos	80
5.6.3. Modificar la estructura de una tabla	81
5.6.4. Ejercicios propuestos sobre modificación de estructura de tablas	82
5.7. Operaciones matemáticas	82
5.7.1. Operaciones matemáticas	82
5.7.2. Funciones de agregación	83
5.8. Creando una tabla	83
5.9. Cargando datos en una tabla	85
5.10. Recuperando información de una tabla	86

5.10.1. Seleccionando todos los datos	86
5.10.2. Seleccionando registros particulares	87
5.10.3. Seleccionando columnas particulares	89
5.11. Ordenando registros	90
5.12. Cálculos con fechas	92
5.13. Trabajando con valores nulos	94
5.14. Coincidencia de patrones	95
5.15. Conteo de filas	98
5.16. Usando más de una tabla	100
5.16.1. Ejercicios propuestos	102
5.17. Operadores	102
5.17.1. Operador de asignación	102
5.17.2. Operadores lógicos	104
5.17.3. Operador Y	104
5.17.4. Operador O	105
5.17.5. Operador de negación	106
5.18. funciones en bases de datos	106
5.18.1. función left	106
5.18.2. función Upper	107
5.18.3. función substring	107
5.18.4. función Lower	107
5.18.5. función Concat	107
5.19. Actualizar la base de datos	108
5.20. Agrupamiento de Registros y funciones agregadas	108
5.20.1. GROUP BY	108
5.20.2. AVG	109
5.20.3. Count	109
5.20.4. Max, Min	109
5.20.5. StDev, StDevP	110
5.20.6. Sum	110
5.20.7. Var, VarP	110
6. Conceptos básicos	111
6.1. Introducción a la programación Web	111
6.1.1. Entorno	111
6.1.2. Ejercicios propuestos	112
6.1.3. Tecnologías	113
6.2. Instalando LAMP en un servidor Debian	114
6.2.1. Prerrequisitos	114
6.2.2. Instalar Apache y actualizar el cortafuegos	114
6.2.3. Ajuste del cortafuegos para el tráfico web	115
6.2.4. Instalar MySQL	116
6.2.5. Instalar PHP	117
6.2.6. Evaluar el procesamiento de PHP sobre tu servidor web	120
6.2.7. Información predeterminada de PHP para Debian	120

7. Fundamentos de programación con PHP	122
7.1. PHP como lenguaje de programación	122
7.1.1. ¿Qué versión usar de PHP?	123
7.1.2. Ejercicio propuesto	123
7.1.3. Ejecución de un programa PHP	123
7.1.4. Tipos de Datos	124
7.1.5. Cadenas de texto	125
7.1.6. Arreglos	126
7.1.7. Objetos	128
7.1.8. Conversión de Tipos de datos	128
7.2. Variables	129
7.2.1. Variables predefinidas	129
7.2.2. Ámbito de una variable	129
7.2.3. Variables variables	131
7.2.4. Variables de los formularios HTML	131
7.2.5. Constantes	133
7.2.6. Expresiones y operadores	133
7.2.7. Estructuras de Control	136
7.3. Funciones	138
7.3.1. Funciones definidas por el usuario	138
7.3.2. Valores devueltos	139
7.3.3. Funciones variables	139
7.4. Procesamiento de formularios	140
7.4.1. Métodos	140
7.5. Parámetros	140
7.6. Páginas con auto-procesamiento	141
7.7. Formularios adhesivos	142
7.8. Parámetros multivaluados	142
7.9. Parámetros multivaluados adhesivos	143
7.10. Manteniendo el estado	144
7.10.1. Campos ocultos en formularios	144
7.11. Cookies	145
7.11.1. Uso de sesiones	147
7.11.2. Reescritura del URL	148
7.11.3. Ejemplo	148
7.11.4. Ejercicio propuesto	149
7.12. Manejo de archivos	149
7.12.1. Directorios	151
7.12.2. Archivos binarios	151
7.12.3. Archivos de texto	151
7.13. Archivos CSV	152
7.13.1. Ejemplo	153
7.13.2. Ejercicio propuesto	154
7.14. Manipulando XML y JSON	155
7.14.1. Lectura de datos XML	155
7.14.2. Escritura de datos mediante XML	156
7.14.3. Lectura de datos JSON	157

7.14.4. Escritura de datos JSON	158
7.15. Generación de imágenes	158
7.15.1. Definición de colores	158
7.15.2. Primitivas de dibujo	159
7.16. Plantillas	160
7.16.1. Uso básico	160
7.16.2. Condicionales	161
7.17. Servicios Web tipo RESTfull	162
7.17.1. Consulta de datos	163
7.17.2. Paso de parámetros	164
7.17.3. Envío de datos	165
8. Conexión con bases de datos	167
8.1. MySQLi	167
8.1.1. Interfaz orientada a objetos	167
8.1.2. Interfaz procedimental	168
8.2. Métodos GET, POST y ENLACE	169
8.2.1. Ejercicio propuesto	171
8.2.2. Variables \$_REQUEST, \$_GET y \$_POST	171
8.2.3. Ejercicio Resuelto	172
8.2.4. Ejercicio Resuelto	173
8.2.5. ¿Desde donde se recuperan los datos?	174
8.2.6. Datos recibidos	175
8.2.7. Ejercicio propuesto	175
8.2.8. Redefinamos los conceptos	175
8.2.9. Enviando datos por enlace	176
8.2.10. Recuperar datos del formulario	177
8.2.11. Recuperar variables POST	178
8.2.12. Recuperar datos de formulario REQUEST	179
8.2.13. Recuperar variables con REQUEST	180
8.2.14. Ejercicio Resuelto	182
8.3. PHP con MySQL	185
8.3.1. Función mysqli_connect	186
8.3.2. Función mysqli_close	188
8.4. Consultas que no devuelven resultado	188
8.5. Recuperar resultados: DATA SEEK, FETCH_ARRAY (MYSQL_RESULT)	189
8.5.1. mysql_result	190
8.5.2. Ejemplo de las funciones anteriores	190
8.5.3. Ejercicio propuesto	193
8.6. Formulario para cargar datos	193
8.6.1. Recibiendo datos del formulario	194
8.6.2. Listar datos estáticos	195
8.6.3. Listar datos dinámicos	196
8.6.4. Agregando enlaces	198
8.6.5. Actualizando o editando datos	199
8.6.6. Recibiendo datos para editar	201
8.6.7. Eliminando datos	202

8.6.8.	Probando y descargando	203
8.6.9.	Consultas	204
8.6.10.	Otras funciones útiles de PDO	209
8.6.11.	Transacciones	211
9.	Bases de datos con PDO	214
9.1.	Introducción	214
9.1.1.	Especificar el tratamiento de errores	215
9.1.2.	Conectar a una base de datos con PDO	216
9.1.3.	Excepciones y opciones con PDO	217
9.1.4.	Registrar datos con PDO	218
9.1.5.	Diferencia entre bindParam() y bindValue()	220
9.1.6.	Consultar datos con PDO	220
9.1.7.	Diferencia entre query() y prepare()/execute()	223
9.1.8.	Otras funciones de utilidad	223
9.1.9.	Transacciones con PDO	224
9.1.10.	sentencias preparadas	225
9.1.11.	Recuperación de datos	226
10.	Un caso práctico	228
10.1.	Caso de Estudio: Biblioteca Libraccio	228
10.1.1.	Diseño de la interacción	229

Capítulo 1

Servidores de Aplicaciones WEB

1.1 Servidores

La conexión entre computadoras y las comunicaciones de alta velocidad han facilitado que los recursos a utilizar **NO** estén en el mismo sitio geográfico que el usuario.

UNIX (y por supuesto GNU/Linux) es el máximo exponente de esta filosofía, ya que desde su inicio ha fomentado el intercambio de recursos y la independencia de dispositivos. Esta filosofía se ha plasmado en algo común como son los servicios.

Un servicio es un recurso (universal o no) que, bajo ciertas condiciones, obtiene información, comparte datos o los procesa a distancia.

Una aplicación web trabaja con servicios para funcionar en una red. Generalmente, dentro de esta red habrá una computadora (o varias, según las configuraciones) que hará el intercambio de información entre las demás. Estas computadoras se denominan servidores y contienen un conjunto de programas para que la información esté centralizada y sea accesible.

Estos servicios reducen de costos y amplían la disponibilidad de la información, pero un servicio centralizado presenta inconvenientes, ya que puede quedar fuera de operación y dejar sin atención a los usuarios.

Los servicios se clasifican como: de vinculación computador-computador o de relación hombre-computador. En el primer caso, son servicios requeridos por otros computadores, en el segundo, son servicios requeridos por los usuarios (hay servicios que actúan en ambas categorías).

En el primer tipo se encuentran el servicio de nombres, Domain Name System (DNS), el servicio de información de usuarios (NIS-YP), el directorio de información LDAP o los servicios de almacenamiento intermedio (proxies).

En la segunda categoría están los servicios de conexión interactiva y ejecución remota (ssh, telnet), transferencia de archivos (ftp), intercambio de información a nivel de usuario, como el correo electrónico (MTA, IMAP, POP), news, World Wide Web, Wiki y archivos (NFS).

1. Proceso servidor:

- es pasivo.
- espera las peticiones de los clientes.

2. Proceso cliente:

- es activo.

- envía las solicitudes de conexión.
- recibe las respuestas del servidor.

1.2 Servidores web

Los servidores web reciben las peticiones de páginas o elementos de la web a través del protocolo http. Es un software alojado en un computador servidor.

Por lo general, es el navegador el que solicita al servidor web la petición del usuario, para finalmente recibir dicho recurso (si fue válida la petición) y traducirla si es necesario a su forma legible por el usuario (la traducción de HTML la hace el navegador).

1.3 Servidores de aplicaciones web

Es una ampliación del punto anterior. Son servidores web que almacenan y gestionan aplicaciones web; ésta es un servicio al que los usuarios acceden a través de la web.

Este tipo de servidores, además de atender peticiones http, son capaces de entender instrucciones de lenguajes avanzados de la web y traducirlas o acceden a recursos de otros servidores. Ese proceso es transparente al usuario. Éste, solicita el servicio a través de su navegador, el servidor de aplicaciones atiende la petición e interpreta el código de la aplicación a fin de traducirla y mostrar al usuario el resultado en su navegador (en formato HTML).

A la forma de trabajar de un servidor de aplicaciones WEB, se le conoce como arquitectura de tres capas (a veces se habla de más capas).

Una primera capa es la del navegador que es capaz de traducir código del cliente (HTML, JavaScript, CSS, Flash, ...).

Para ello esa capa debe de disponer de todos los componentes necesarios para hacer esa labor en el computador del usuario.

La segunda capa, el servidor de aplicaciones, que traduce código en el lado del servidor (JSP, PHP, Ruby on Rails, Cold Fussion...) y lo convierte al formato entendible por el navegador. La tercera capa son todos los servicios a los que accede el servidor de aplicaciones para realizar la tarea encomendada a la aplicación (como el acceso a la base de datos), vea la figura 1.1 :

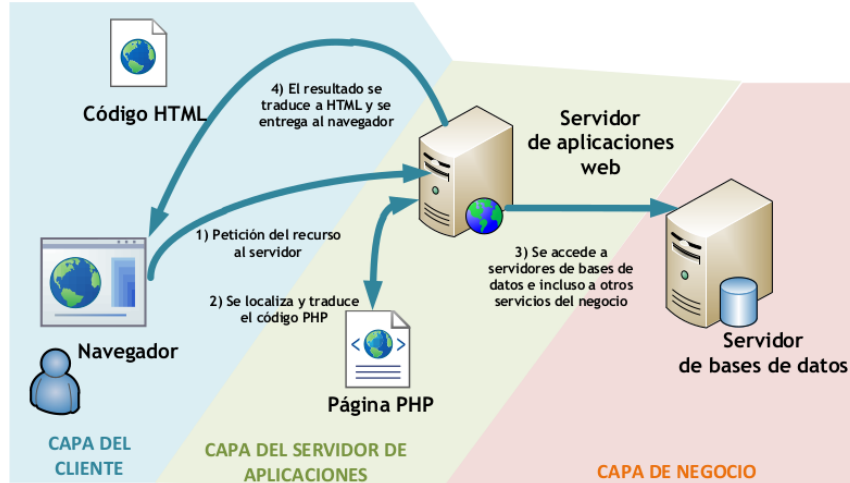


Figura 1.1: Esquema de una aplicación WEB.

1.4 Tecnologías para crear aplicaciones web

Estás son: apache, php, mariadb, html, css, JavaScript, bootstrap

1.4.1 Apache

Apache es uno de los servidores más populares y con mayores prestaciones de HTTP (Hyper-Text Transfer Protocol), tiene un diseño modular y soporta extensiones dinámicas de módulos durante su ejecución. Es muy configurable en cuanto al número de servidores y de módulos disponibles y soporta diversos mecanismos de autenticación, control de acceso, metafilas, proxy caching, servidores virtuales, etc. Con módulos (incluidos en Debian) es posible tener PHP7, Perl, Java Servlets, SSL y otras extensiones.

Apache está diseñado para ejecutarse como un proceso daemon standalone. En esta forma, crea un conjunto de procesos hijos que gestionarán las peticiones de entrada. También puede ejecutarse como Internet daemon a través de inetd, por lo que se pondrá en marcha cada vez que se reciba una petición pero no es recomendado.

Además, se trata de un software de código abierto que utiliza una licencia de tipo Apache License que es una variante de la licencia GPL de Linux. Eso significa que se puede distribuir sin problemas e incluso mejorar. Dispone de multitud de módulos que convierten a Apache en un servidor capaz de gestionar todo tipo de aplicaciones, lo que también le convierte en el servidor de aplicaciones más popular de la actualidad; por ejemplo dispone de módulos para:

- Implementar SSL. Protocolo de seguridad en la transferencia de información.
- Enlace con el servidor Tomcat para implementar aplicaciones JSP.
- Módulo para Perl.
- Módulo para PHP.
- Módulo para Python.
- etc.

1.4.2 NGINX

Pronunciado como **engine-ex**, es un servidor web de código abierto que, desde su éxito inicial como servidor web, ahora también es usado como proxy inverso, cache de HTTP, y balanceador de carga.

NGINX está diseñado para ofrecer un bajo uso de memoria y alta concurrencia.

En lugar de crear nuevos procesos para cada solicitud web, NGINX usa un enfoque asíncronico basado en eventos donde las solicitudes se manejan en un solo hilo (single-thread). Por otro lado, una alternativa como el servidor web Apache crearía un hilo separado para cada proceso.

Con NGINX, un proceso maestro puede controlar múltiples procesos de trabajo y mantiene los procesos de trabajo, los cuales hacen el procesamiento real.

Debido a que NGINX es asíncrono, cada solicitud se ejecuta por el proceso de trabajo de forma concurrente sin bloquear otras solicitudes.

NGINX es muy ligero, sirve como solución al problema de rendimiento de manejar 100,000 conexiones concurrentes. Se puede ampliar con módulos para servir aplicaciones web.

1.4.3 PHP (Personal Home Pages)

Se trata de un lenguaje de scripts de servidor; es decir código que se incrusta en las páginas HTML y que requiere ser traducido por un servidor de aplicaciones que devolverá un resultado en formato HTML. PHP significa Hypertext Pre Processor y se trata del lenguaje de scripts de servidor más popular.

Un lenguaje de scripts, en general, es un lenguaje cuyo código se incrusta dentro de otro. Es el caso de JavaScript que es un lenguaje que va incrustado dentro del código HTML de una página web. Pero, en el caso de JavaScript está en el lado del cliente; es decir es el navegador de Internet el que tiene que tener la capacidad de interpretar el código del lenguaje script, además del HTML. Eso provoca una desventaja: los navegadores tienen que tener capacidades añadidas y un nuevo lenguaje de ese tipo implica nuevos plugin para los navegadores, con el riesgo de que algunos usuarios naveguen por las páginas web sin verlas adecuadamente porque su navegador no tiene instalado el plugin.

Por ello los lenguajes de script de servidor han tenido mucho éxito. En ese caso, es el servidor el que interpreta el lenguaje script y devuelve al navegador el resultado de interpretar dicho lenguaje, que siempre es HTML.

PHP es gratuito y software de código abierto que tiene una relación excelente con Apache, MySQL y Linux; aunque actualmente en Windows también se instala muchísimo. PHP se puede instalar también en servidores IIS de Microsoft y en otros muchos; además puede utilizar sistemas de bases de datos como Oracle, Informix, DB2, ...

1.4.4 Contraseñas de usuarios

Hay que asegurar desde el primer momento que la instalación de MySQL que los usuarios NO sobrepasen sus privilegios y dañar las bases de datos. Por ello hay que asegurar que todos los usuarios (y en especial los administrativos) tienen contraseña.

En Windows hay un usuario root (superusuario) que tienen privilegios totales pero que sólo accede a la computadora local. Para permitir el acceso con esos privilegios desde otra computadora, se crea otro usuario root (durante la instalación de MySQL se pregunta esa posibilidad).

En Linux, los usuarios root permiten el acceso local.

Hay cuentas anónimas (no tienen nombre de usuario) algunas para acceder de forma local y otras no. Para asegurar el acceso se deben poner contraseñas a las cuentas anónimas o bien eliminarlas.

1.4.5 Mariadb

Las bases de datos son el método preferido para el almacenamiento estructurado de datos. Desde las grandes aplicaciones multiusuario, hasta los teléfonos móviles y las agendas electrónicas utilizan tecnología de bases de datos para asegurar la integridad de los datos y facilitar la labor tanto de usuarios como de los programadores que las desarrollaron.

El diseño de bases de datos tiene también un capítulo dedicado a aprender a modelar y representar gráficamente una base de datos, a detectar los posibles problemas de diseño antes de que éstos afecten a la aplicación, y a construir bases de datos óptimas para los distintos casos de relaciones entre entidades que formarán nuestra base de datos.

MySQL es un sistema gestor de bases de datos (SGBD, DBMS) muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Aunque carece de algunas características avanzadas disponibles en otros SGBD del mercado, es una opción atractiva tanto para aplicaciones comerciales, como de entretenimiento precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo licencia GPL le otorgan como beneficios adicionales (no menos importantes) contar con un alto grado de estabilidad y un rápido desarrollo.

MySQL está disponible para múltiples plataformas, la seleccionada para los ejemplos de este libro es GNU/Linux. Sin embargo, las diferencias con cualquier otra plataforma son prácticamente nulas, ya que la herramienta utilizada en este caso es el cliente `mysql-client`, que permite interactuar con un servidor MySQL (local o remoto) en modo texto. De este modo es posible realizar todos los ejercicios sobre un servidor instalado localmente o, a través de Internet, sobre un servidor remoto.

Para la realización de todas las actividades, es imprescindible que dispongamos de los datos de acceso del usuario administrador de la base de datos. Aunque en algunos de ellos los privilegios necesarios serán menores, para los capítulos que tratan la administración del SGBD será imprescindible disponer de las credenciales de administrador.

1.4.6 html5

HTML5 provee básicamente tres características: estructura, estilo y funcionalidad. Nunca fue declarado oficialmente pero, incluso cuando algunas API (Interface de Programación de Aplicaciones) y la especificación de CSS3 por completo no son parte del mismo, HTML5 es considerado el producto de la combinación de HTML, CSS y JavaScript. Estas tecnologías son altamente dependientes y actúan como una sola unidad organizada bajo la especificación de HTML5. a cargo de la estructura, CSS presenta esa estructura y su contenido en la pantalla y JavaScript hace el resto que (como veremos más adelante) es extremadamente significativo.

Más allá de esta integración, la estructura sigue siendo parte esencial de un documento. La misma provee los elementos necesarios para ubicar contenido estático o dinámico, y es también una plataforma básica para aplicaciones. Con la variedad de dispositivos para acceder a Internet y la diversidad de interfaces disponibles para interactuar con la web, un aspecto básico como la estructura se vuelve parte vital del documento. Ahora la estructura debe proveer forma, organi-

zación y flexibilidad, y debe ser tan fuerte como los fundamentos de un edificio.

Para trabajar y crear sitios webs y aplicaciones con HTML5, necesitamos saber primero cómo esa estructura es construida. Crear fundamentos fuertes nos ayudará más adelante a aplicar el resto de los componentes para aprovechar completamente estas nuevas tecnologías.

1.4.7 ¿Cómo funciona el XHTML?

HTML y XHTML son lenguajes basados en etiquetas (tags). Una etiqueta tiene la siguiente forma: `<etiqueta>Algo aquí dentro</etiqueta>` Volviendo al ejemplo anterior: la etiqueta para el título principal en la página es `<h1>`.

Entonces quedaría: `<h1>Viste la película Matrix?</h1>`

Como ve, `<h1>` marca el inicio de la etiqueta, mientras que `</h1>` se encarga de cerrarla. Hay etiquetas que funcionan con una sola parte, y son así: `<etiqueta />` Observe el espacio en blanco que hay entre el nombre de la etiqueta y la barra /. Es muy importante para que los navegadores antiguos no se vuelvan locos.

Hay etiquetas que pueden modificarse con atributos. Ahora mismo no hace falta entender qué hacen, ya los veremos más adelante. Se escriben de la siguiente forma: `<etiqueta atributo="valor">`

1.4.8 Css3

CSS es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados "documentos semánticos"). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.

Al crear una página web, se utiliza en primer lugar el lenguaje HTML/XHTML para marcar los contenidos, es decir, para designar la función de cada elemento dentro de la página: párrafo, titular, texto destacado, tabla, lista de elementos, etc.

Una vez creados los contenidos, se utiliza CSS para definir el aspecto de cada elemento: color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página, etc.

Las hojas de estilos aparecieron poco después que el lenguaje de etiquetas SGML, alrededor del año 1970. Desde la creación de SGML, se observó la necesidad de definir un mecanismo que permitiera aplicar de forma consistente diferentes estilos a los documentos electrónicos.

El gran impulso de los lenguajes de hojas de estilos se produjo con el boom de Internet y del lenguaje HTML para la creación de documentos electrónicos con la misma apariencia en diferentes navegadores. El organismo W3C (<http://www.w3.org/>) (World Wide Web Consortium), es el encargado de crear todos los estándares relacionados con la web.

La propuesta CHSS fue realizada por Håkon Wium Lie, SSP, Lie y Bos fue llamada CSS (Cascading Style Sheets). A finales de 1996, el W3C publicó la primera recomendación oficial, conocida como "CSS nivel 1".

A principios de 1997, el W3C decide separar los trabajos del grupo de HTML en tres secciones: el grupo de trabajo de HTML, el grupo de trabajo de DOM y el grupo de trabajo de CSS.

El 12 de Mayo de 1998, la segunda recomendación es "CSS nivel 2". En 1998, la siguiente recomendación de CSS, conocida como "CSS nivel 3", continúa en desarrollo y producción.

El trabajo del diseñador web siempre está limitado por los navegadores que utilizan los usuarios para acceder a sus páginas. Por este motivo es imprescindible conocer el soporte de CSS en cada uno de los navegadores más utilizados del mercado.

Internamente, los navegadores están divididos en varios componentes. La parte del navegador que se encarga de interpretar el código HTML y CSS para mostrar las páginas se denomina motor. Desde el punto de vista del diseñador CSS, la versión de un motor es mucho más importante que la versión del propio navegador.

La siguiente tabla muestra el soporte de CSS 1, CSS 2.1 y CSS 3 de los cinco navegadores más utilizados por los usuarios:

Navegador	Motor	CSS 1	CSS 2.1	CSS 3
Internet Explorer	Trident	Completo desde la versión 6.0	Completo desde la versión 8.0	Prácticamente nulo
Firefox	Gecko	Completo	Completo	Selectores, pseudo-clases y algunas propiedades
Safari	WebKit	Completo	Completo	Todos los selectores, pseudo-clases y muchas propiedades
Opera	Presto	Completo	Completo	Todos los selectores, pseudo-clases y muchas propiedades
Google Chrome	WebKit	Completo	Completo	Todos los selectores, pseudo-clases y muchas propiedades

Tabla 1.1: Algunos navegadores WEB.

1.4.9 JavaScript

JavaScript es un lenguaje de "scripting"(guiones o rutinas) y no puede hacer un programa con él sino sólo trabajar con páginas web. Javascript **NO** se instala, puede crear un script de JavaScript con cualquier editor de texto o web.

JavaScript sirve principalmente para la gestión de la interfaz cliente/servidor, insertado en un documento HTML reconoce y trata en el lado del cliente, los eventos generados por el usuario. Estos eventos son el recorrido del propio documento HTML o la gestión de un formulario.

Cuando la página HTML es un formulario que accede a un directorio telefónico, se puede insertar un script que verifique la validez de los parámetros proporcionados por el usuario. Esta prueba se efectúa localmente y no necesita acceso a la red.

Por otro lado, utilizará JavaScript para efectuar varias operaciones a la vez; como acompañar el acceso a un documento HTML de la visualización de un vídeo o la ejecución de un Applet de Java etc.

Para saber si un navegador soporta JavaScript independientemente de su tipo o versión y avisarle al visitante el por qué la página no se muestra de forma correcta, aplique el siguiente código.

Primero, creemos un HTML sencillo para incluir nuestro JavaScript y verificar nuestras validaciones, con el siguiente contenido:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <script type="text/javascript">
5 </script>
6 <meta charset="UTF-8">
7 <title>Detectando el navegador del usuario</title>
8 </head>
9 <body>
10 </body>
11 </html>
```

Ya con nuestro código HTML insertamos código JavaScript dentro de las etiquetas `<head>`. Primero, detectemos el navegador más utilizado, Google Chrome, utilizando el objeto `navigator`:

```
1 var es_chrome = navigator.userAgent.toLowerCase().indexOf('chrome') >
  -1;
2 if(es_chrome){
3     alert("El navegador que está utilizando es Chrome");
4 }
```

Ahora, veamos la manera de hacerlo para Firefox:

```
1 var es_firefox = navigator.userAgent.toLowerCase().indexOf('firefox')
  > -1;
2 if(es_firefox){
3     alert("El navegador que está utilizando es Firefox");
4 }
```

1.4.10 Bootstrap

Bootstrap es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. A diferencia de muchos frameworks web, sólo se ocupa del desarrollo front-end.

Bootstrap es usado por la NASA y la MSNBC entre otras organizaciones; además, es el segundo proyecto más destacado en GitHub¹.

Características

Bootstrap tiene un soporte relativamente incompleto para HTML5 y CSS 3, pero es compatible con la mayoría de los navegadores web. La información básica de compatibilidad de sitios web o aplicaciones está disponible para todos los dispositivos y navegadores. Existe un concepto de compatibilidad parcial que hace disponible la información básica de un sitio web para todos los dispositivos y navegadores. Por ejemplo, las propiedades introducidas en CSS3 para las esquinas redondeadas, gradientes y sombras son usadas por Bootstrap a pesar de la falta de soporte de navegadores antiguos. Esto extiende la funcionalidad de la herramienta, pero no es requerida para

¹GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git

su uso.

Desde la versión 2.0 soporta diseños web adaptables. Esto significa que el diseño gráfico de la página se ajusta dinámicamente, tomando en cuenta las características del dispositivo usado (Computadoras, tabletas, teléfonos móviles).

Bootstrap es de código abierto y está disponible en GitHub. Los desarrolladores están motivados a participar en el proyecto y a hacer sus propias contribuciones a la plataforma.

Bootstrap es modular y consiste esencialmente en una serie de hojas de estilo LESS que implementan la variedad de componentes de la herramienta. Una hoja de estilo llamada bootstrap.less incluye los componentes de las hojas de estilo. Los desarrolladores pueden adaptar el mismo archivo de Bootstrap, seleccionando los componentes que deseen usar en su proyecto.

Los ajustes son posibles en una medida limitada a través de una hoja de estilo de configuración central. Los cambios más profundos son posibles mediante las declaraciones LESS.

El uso del lenguaje de hojas de estilo LESS permite el uso de variables, funciones y operadores, selectores anidados, así como clases mixin.

Desde la versión 2.0, la configuración de Bootstrap también tiene una opción especial de "Personalizar".^{en} la documentación. Por otra parte, los desarrolladores eligen en un formulario los componentes y ajustes deseados, y de ser necesario, los valores de varias opciones a sus necesidades.

El paquete consecuentemente generado ya incluye la hoja de estilo CSS precompilada.

El siguiente ejemplo ilustra como funciona. El código HTML define un simple formulario de búsqueda y una lista de resultados en un formulario tabular. La página consiste en elementos regulares y semánticos de HTML 5, y alguna información adicional de la clase de CSS de acuerdo con la documentación de Bootstrap. La figura muestra la representación del documento en Mozilla Firefox.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Ejemplo de Bootstrap</title>
5  <!-- Bootstrap CSS -->
6  <link rel="css/bootstrap.min.css">
7  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
8  </head>
9  <body>
10 <div class="container">
11 <h1>Search</h1>
12 <label>Formulario sencillo de búsqueda.</label>
13 <!-- Formulario de búsqueda con un campo de entrada (input) y un botón -->
14 <form class="well form-search">
15 <input type="text" class="input-medium search-query">
16 <button type="submit" class="btn btn-primary">Buscar</button>
17 </form>
18 <h2>Results</h2>
19 <!-- Tabla con celdas de color de fondo alternantes y con marco -->
20 <table class="table table-striped table-bordered">
21 <thead>
22 <tr>
23 <th>#</th>
24 <th>Título</th>
```

```
25 </tr>
26 </thead>
27 <tbody>
28 <tr>
29 <td>1</td>
30 <td>Lorem ipsum dolor sit amet</td>
31 </tr>
32 <tr>
33 <td>2</td>
34 <td>Consetetur sadipscing elitr</td>
35 </tr>
36 <tr>
37 <td>3</td>
38 <td>At vero eos et accusam</td>
39 </tr>
40 </tbody>
41 </table>
42 </div>
43
44 <!-- jQuery -->
45 <script src="js/jquery-2.2.4.min.js"></script>
46 <!-- Bootstrap JS -->
47 <script src="js/bootstrap.min.js"></script>
48 </body>
49 </html>
```

Capítulo 2

Primeros ejemplos

2.1 html5

En este capítulo aprenderemos cómo construir una plantilla para futuros proyectos usando los nuevos elementos HTML introducidos en las herramientas HTML5, CSS3 y Javascript. Empecemos por lo básico, paso a paso.

2.1.1 Editor de texto

Puedes trabajar con tu editor de texto preferido, algunos editores son: gedit, kate, geany, nano, netbeans, eclipse, dreamweaver, php storm, bluefish, quanta plus, sblime text, atom, brackets, visual studio code, entre otros. Unos editores son software libre y otros son pago con licenciamiento; también consideremos si el editor es multiplataforma y puede utilizarse desde Windows, GNU/Linux o Mac OS X (10.7+). Igualmente, estamos atentos a los requerimientos de hardware de cada editor.

Lo importante es que nuestro editor aporte características útiles a la hora de programar o editar código. Si el editor seleccionado tiene funcionalidades útiles y cómodas desde el punto de la usabilidad y eficiencia convertirá nuestro trabajo de edición de texto en una experiencia sencilla y agradable, a medida que aprendemos utilizarlo.

2.1.2 Ruta o Árbol de navegación

Una aplicación WEB debe estar alojada en la siguientes ruta en Debian y derivados: /var

/www

/html

/miaplicacion

/imagenes

/js

/css

/funciones

sus programas

Recuerda que el programa principal debe llamarse index.php.

Considera la permisología de la carpeta (miaplicacion), para ello ejecuta en la terminal, como

root:

```
chmod -R 0775 /var/www/html/miaplicacion
```

y el permiso para navegación por usuarios anónimos:

```
chown -R :www-data /var/www/html/miaplicacion
```

Ahora, empecemos a programar y a ejecutar nuestras aplicaciones WEB.

2.2 Programa Hola, mundo

En primer lugar, indiquemos el tipo de documento creado:

```
<!DOCTYPE html>
```

IMPORTANTE: Es la primera línea del archivo, sin espacios o líneas que la precedan. De esta forma, el modo estándar del navegador es activado e integra elementos básicos de HTML5 siempre que sea posible, o sean ignorados.

A continuación un programa HTML5 sencillo:

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale
  =1.0">
6 <meta http-equiv="X-UA-Compatible" content="ie=edge">
7 <title>Document</title>
8 </head>
9 <body>
10 Hola, mundo
11 </body>
12 </html>
```

ejecutemos así, en el cintillo de su navegador: <http://localhost/miaplicación>:

2.3 Funcionamiento básico de CSS

El tanto el ejemplo anterior como el siguiente muestra una página HTML con estilos definidos sin utilizar CSS:

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale
  =1.0">
6 <meta http-equiv="X-UA-Compatible" content="ie=edge">
7 <title>Ejemplo de estilos sin CSS</title>
8 </head>
9 <body>
10 <h1><font color="red" face="Arial" size="5">Titular de la pagina</
  font></h1>
11 <p><font color="gray" face="Verdana" size="2">Un parrafo de texto no
  muy
```

```

12 | largo.</font></p>
13 | </body>
14 | </html>

```

El ejemplo anterior utiliza la etiqueta `` con sus atributos `color`, `face` y `size` para definir color, tipografía y tamaño del texto de cada elemento del documento.

El principal problema es definir el aspecto de los elementos se ven.

Si la página tuviera 50 elementos diferentes, habría que insertar 50 etiquetas ``. Si el sitio web entero se compone de 10.000 páginas diferentes, habría que definir 500.000 etiquetas ``. Como cada etiqueta `` tiene 3 atributos, habría que definir 1.5 millones de atributos.

Por otra parte, el diseño de los sitios web está en constante evolución y es habitual modificar cada cierto tiempo los colores principales de las páginas o la tipografía utilizada para el texto. Si emplea la etiqueta ``, habría que modificar el valor de 1.5 millones de atributos para cambiar el diseño general del sitio web.

La solución que propone CSS es la siguiente:

```

1 | <!DOCTYPE html>
2 | <html lang="es">
3 | <head>
4 | <meta charset="UTF-8">
5 | <meta name="viewport" content="width=device-width, initial-scale
   |     =1.0">
6 | <meta http-equiv="X-UA-Compatible" content="ie=edge">
7 | <style type="text/css">
8 | h1 { color: red; font-family: Arial;
9 | font-size: large; }
10 | p { color: gray; font-family: Verdana; font-size: medium; }
11 | </style>
12 | </head>
13 | <body>
14 | <h1>Titular de la pagina</h1>
15 | <p>Un parrafo de texto no muy largo.</p>
16 | </body>
17 | </html>

```

CSS separa los contenidos de la página y su aspecto o presentación. En el ejemplo anterior, dentro de la propia página HTML se reserva una zona en la que se incluye la información relacionada con los estilos de la página. Cómo incluir CSS en un documento HTML

Una de las principales características de CSS es su flexibilidad y las diferentes opciones que ofrece para realizar una misma tarea. De hecho, existen tres opciones para incluir CSS en un documento HTML.

2.3.1 Incluir CSS en el mismo documento HTML

Los estilos se definen en una zona específica del propio documento HTML. Se emplea la etiqueta `<style>` de HTML y solamente se pueden incluir en la cabecera del documento (sólo dentro de la sección `<head>`).

Veamos el siguiente ejemplo:

```

1 | <!DOCTYPE html>
2 | <html lang="en">

```

```

3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=iso
  -8859-1" />
5 <title>Ejemplo de estilos CSS en el propio documento</title>
6 <style type="text/css">
7 p { color: black; font-family: Verdana; }
8 </style>
9 </head>
10 <body>
11 <p>Un parrafo de texto.</p>
12 </body>
13 </html>

```

Este método se emplea cuando se define un número pequeño de estilos o cuando se quieren incluir estilos específicos en una determinada página HTML que completen los estilos que se incluyen por defecto en todas las páginas del sitio web.

El principal inconveniente es que si se quiere hacer una modificación en los estilos definidos, es necesario modificar todas las páginas que incluyen el estilo que se va a modificar.

2.3.2 Definir CSS en un archivo externo

En este caso, todos los estilos CSS se incluyen en un archivo de tipo CSS que las páginas HTML enlazan mediante la etiqueta `<link>`. Un archivo de tipo CSS no es más que un archivo simple de texto cuya extensión es `.css`. Se pueden crear todos los archivos CSS que sean necesarios y cada página HTML puede enlazar tantos archivos CSS como necesite.

Si queremos incluir los estilos del ejemplo anterior en un archivo CSS externo, hacemos los siguientes pasos:

1. Se crea un archivo de texto y se le añade solamente el siguiente contenido:
 - `p color: black; font-family: Verdana;`
 - Se guarda el archivo de texto con el nombre `estilos.css`. Se debe poner especial atención a que el archivo tenga extensión `.css` y no `.txt`
 - En la página HTML se enlaza el archivo CSS externo mediante la etiqueta `<link>`.

Veamos el siguiente ejemplo:

```

1 <!DOCTYPE>
2 <html lang="es">
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=iso
  -8859-1" />
5 <title>Ejemplo de estilos CSS en un archivo externo</title>
6 <link rel="stylesheet" type="text/css" href="/css/estilos.css" media
  ="screen" />
7 </head>
8 <body>
9 <p>Un parrafo de texto.</p>
10 </body>
11 </html>

```


Cuando el navegador carga la página HTML anterior, previo a mostrar sus contenidos también descarga los archivos CSS externos enlazados mediante la etiqueta `<link>` y aplica los estilos a los contenidos de la página. La etiqueta `<link>` incluye cuatro atributos cuando se enlaza un archivo CSS:

- `rel` : indica el tipo de relación que tiene el recurso enlazado (en este caso, el archivo CSS) y la página HTML. Para los archivos CSS, siempre se utiliza el valor `stylesheet`.
- `type` : indica el tipo de recurso enlazado. Sus valores son un estándar y para los archivos CSS su valor siempre es `text/css`.
- `href` : indica la URL del archivo CSS que contiene los estilos. La URL indicada puede ser relativa o absoluta y apuntar a un recurso interno o externo al sitio web.
- `media` : indica el medio en el que se van a aplicar los estilos del archivo CSS. Más adelante se explican en detalle los medios CSS y su funcionamiento.

De todas las formas de incluir CSS en las páginas HTML, esta es la más utilizada con mucha diferencia. La principal ventaja es que se puede incluir un mismo archivo CSS en multitud de páginas HTML, por lo que se garantiza la aplicación homogénea de los mismos estilos a todas las páginas que forman un sitio web.

Con este método, el mantenimiento del sitio web se simplifica al máximo, ya que un sólo cambio en un único archivo CSS varía de forma instantánea los estilos de las páginas HTML que enlazan ese archivo.

La etiqueta `<link>` vincula los archivos CSS externos; mientras que la etiqueta `<style>` empotra el código. La forma alternativa de incluir un archivo CSS externo se muestra a continuación:

```
1 |<!DOCTYPE>
2 |<head lang='en'>
3 |<meta http-equiv="Content-Type" content="text/html; charset=iso
  |-8859-1" />
4 |<title>Ejemplo de estilos CSS en un archivo externo</title>
5 |<style type="text/css" media="screen">
6 |@import '/css/estilos.css';
7 |</style>
8 |</head>
9 |<body>
10|<p>Un parrafo de texto.</p>
11|</body>
12|</html>
```

En este caso, para incluir en HTML los estilos definidos en archivos CSS externos, se utiliza una regla especial de tipo `@import`. Las reglas `@import` siempre preceden a cualquier otra regla CSS (con la única excepción de la regla `@charset`).

La URL del archivo CSS externo se indica mediante una cadena de texto encerrada con comillas simples o dobles o mediante la palabra reservada `url()`. De esta forma, las siguientes reglas `@import` son equivalentes:

```
1 |@import '/css/estilos.css';
2 |@import "/css/estilos.css";
3 |@import url('/css/estilos.css');
4 |@import url("/css/estilos.css");
```

2.3.3 Incluir CSS en los elementos HTML

El último método para incluir estilos CSS en HTML es el peor y el menos utilizado, con los mismos problemas de la utilización de las etiquetas ``.

Veamos el siguiente ejemplo:

```
1 <!DOCTYPE>
2 <html lang="es">
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=iso
  -8859-1" />
5 <title>Ejemplo de estilos CSS en el propio documento</title>
6 </head>
7 <body>
8 <p style="color: black; font-family: Verdana;">Un parrafo de texto.</
  p>
9 </body>
10 </html>
```

Esta forma de incluir CSS directamente en los elementos HTML se utiliza en determinadas situaciones en las que se debe incluir un estilo muy específico para un sólo elemento concreto.

2.4 Javascript

```
1 <!DOCTYPE>
2 <html lang='es'>
3 <head>
4 <title></title>
5 <meta name="generator" content="Bluefish 2.0.2" >
6 <meta name="author" content="jodocha" >
7 <meta name="date" content="2011-10-12T09:23:01-0430" >
8 <meta name="ROBOTS" content="NOINDEX, NOFOLLOW">
9 <meta http-equiv="content-type" content="text/html; charset=UTF-8">
10 <meta http-equiv="content-type" content="application/xhtml+xml;
   charset=UTF-8">
11 <meta http-equiv="content-style-type" content="text/css">
12 <meta http-equiv="expires" content="0">
13 <SCRIPT type="text/javascript">
14 <!-- inicio
15 function browsertest () {
16 document.write(" Lo sabia ? Su navegador tiene JavaScript.")
17 }
18 // final -->
19 </SCRIPT>
20 </head>
21 <BODY onload="browsertest();">
22 </BODY>
23 </HTML>
```

2.4.1 JavaScript en documentos HTML

He aquí un código que ilustra la integración directa de código JavaScript en un documento HTML:

```

1 <HTML>
2 <HEAD>
3 <TITLE>Primer ejemplo de JavaScript</TITLE>
4 </HEAD>
5 <BODY>
6 Esto es texto normal de un documento HTML
7 <SCRIPT LANGUAGE="JavaScript"> document.write("Texto generado desde
8 JavaScript")
9 </SCRIPT>
10 Esto es, de nuevo, HTML
11 </body>
12 </HTML>

```

Como se observa, este código tiene la apariencia de un HTML estándar. La novedad viene dada por la presencia del fragmento correspondiente al código JavaScript:

```

1 <SCRIPT LANGUAGE="JavaScript"> document.write('Texto generado desde
2 Javascript")
3 </SCRIPT>

```

Para ver el resultado de su ejecución, basta con cargar dicho documento con cualquiera de los clientes Web mencionados. La salida (para ambos clientes Web) se compone de tres líneas de texto:

- Esto es texto normal de un documento HTML.
- Texto generado desde JavaScript.

2.5 De nuevo HTML

En realidad no se trata de un script útil, lo que ofrece (mostrar una línea de texto) se hacer directamente en HTML y, sin duda, con mayor comodidad. Sólo demuestra el funcionamiento del código `<SCRIPT>`.

En efecto, cualquier sentencia delimitada por las etiquetas `<SCRIPT>` y `</SCRIPT>` se considera código JavaScript.

En este caso particular, hemos utilizado `document.write`, una de las funciones importantes de JavaScript, que escribe algo en el documento actual (el documento HTML que contiene el ejemplo).

2.6 Archivos con código JavaScript

El atributo `SRC` del código `SCRIPT` del lenguaje HTML especifica un archivo con el código JavaScript (sin incrustar el código JavaScript en el documento HTML).

Por ejemplo:

```

1 <HEAD>
2 <SCRIPT SRC="comun.js"> ...
3 </SCRIPT>
4 </HEAD>
5 <BODY> ...

```

Este atributo es especialmente útil para compartir funciones entre numerosos documentos HTML. Las sentencias JavaScript del interior de un código `<SCRIPT SRC= ... >` se ignoran a menos que la inclusión cause un error.

Se incluye una sentencia que muestre un mensaje de error en caso de no cargar el archivo de código.

2.7 Sin sentencia HTML

Clientes que no soportan JavaScript.- Estos clientes no admiten la etiqueta HTML `<script>`. Consideran la etiqueta `SCRIPT` y su contenido como texto normal, por ello, se hace preciso ocultar el código a clientes que no lo soporten. Para evitar esto, se utilizan los comentarios de HTML entre las etiquetas `<SCRIPT>` `</SCRIPT>`:

```
1 |<SCRIPT LANGUAGE="JavaScript">
2 |</SCRIPT>
```

Recordemos que es la forma de insertar comentarios en HTML.

Otra forma de conocer si un cliente soporta JavaScript es insertar el código `<NOSCRIPT>...</NOSCRIPT>`. De modo, los navegadores que no soporten JavaScript ejecutan las sentencias HTML alternativas incluidas dentro de esta etiqueta.

2.8 Bootstrap

2.8.1 Sistema de cuadrilla y diseño sensible

Bootstrap viene con una disposición de cuadrilla estándar de 940 píxeles de ancho. Alternativamente, el desarrollador puede usar un diseño de ancho-variable. Para ambos casos, la herramienta tiene cuatro variaciones para hacer uso de distintas resoluciones y tipos de dispositivos: teléfonos móviles, formato de retrato y paisaje, tabletas y computadoras con baja y alta resolución (pantalla ancha). Esto ajusta el ancho de las columnas automáticamente.

2.8.2 Entendiendo la hoja de estilo CSS

Bootstrap proporciona un conjunto de hojas de estilo que proveen definiciones básicas para todos los componentes de HTML. Vea la figura 2.1. Esto otorga una uniformidad al navegador y al sistema de anchura, da una apariencia moderna para el formateo de los elementos de texto, tablas y formularios.



Figura 2.1: Etiquetas a usar el diseño HTML5 y Bootstrap.

2.8.3 Creando una cuadrilla de diseño fija

```

1 <div class="row">
2 <div class="col-md-4">...</div>
3 <div class="col-md-8">...</div>
4 </div>

```

2.8.4 Creando una cuadrilla de diseño fija con una cuadrilla de diseño fluida anidada

```

1 <div class="row">
2 <div class="col-md-4">
3 <div class="4">...</div>
4 <div class="4">...</div>
5 <div class="4">...</div>
6 </div>
7 <div class="col-md-8">...</div>
8 </div>

```

Capítulo 3

La programación

3.1 Qué es el HTML?

HTML significa "HyperText Markup Language" es un lenguaje semántico, lo que quiere decir que no define el aspecto de las cosas, sino lo que significan. Por ejemplo, si tiene el título de su página, en lugar de decir "Lo quiero grande en letras rojas", le indicas al navegador que "Este es el título principal de la página". Haz algo para que destaque. Y ese **algo** lo deja a decisión del navegador.

3.2 Y eso de CSS?

CSS son "Cascading Style Sheets". Si el documento HTML está estructurado, cambia totalmente su apariencia sin tocar una sola línea de código en el archivo .html. Es separar el contenido del aspecto, y es de gran importancia.

3.3 Características del lenguaje HTML

Cómo se hace una página Web? Cuando los diseñadores del WWW se hicieron esta pregunta decidieron que se debían cumplir, entre otras, las siguientes características:

- El Web tiene que ser distribuido: la información repartida en páginas no muy grandes enlazadas entre sí.
- El Web tiene que ser hipertexto y debía ser fácil navegar por él.
- Tiene que ser compatible con todo tipo de computadores (PC, Macintosh, estaciones de trabajo...) y con todo tipo de sistemas operativos (Windows, MS-DOS, UNIX, MAC-OS, ...).
- Debe ser dinámico: el proceso de cambiar y actualizar la información debe ser ágil y rápido.

Estas características son las que marcaron el diseño de todos los elementos del WWW incluida la programación de páginas Web. Como respuesta a todos estos requisitos se creó el lenguaje HTML.

Éste será el encargado de convertir un archivo de texto inicial en una página web con diferentes tipos y tamaños de letra, con imágenes, animaciones, formularios interactivos, etc.

Qué necesita para crear una página web? Una de las características de este lenguaje más importantes para el programador es que no necesita algún programa especial para crear una página Web. Gracias a ello se ha conseguido crear páginas con cualquier computador y sistema operativo.

El código HTML es texto puro y por tanto lo único necesario para escribirlo es un editor de texto como el que acompañan a los sistemas operativos: edit en MS-DOS, notepad en Windows, vi y vim, emacs, kate, gedit, nano en UNIX y Linux, etc.

También podemos usar procesadores de texto, que son editores con capacidades añadidas, pero hay que tener cuidado porque en ocasiones hacen traducciones automáticas del código HTML que no siempre son deseadas. En este último caso, que deberemos guardar el archivo en modo texto.

Una vez que hemos escrito el código grabamos el archivo (con formato de texto) con la extensión .php (en minúsculas). Los siguientes son nombres válidos de archivos que contengan código HTML: index.php, principal.php, PRINCIPAL.php, etc.

La página o programa principal, la que se ejecuta al entrar en el sitio web se llama index.php.

Por último, queda comentar un par de reglas de las etiquetas: siempre se escriben en minúsculas y los atributos entre comillas.

3.4 Estructura HTML

Este apartado es muy importante. No es largo, así que presta atención. Ahora aprenderás a formar una plantilla o template de tus archivos para usarlo más adelante.

3.4.1 La codificación

Una línea de un documento HTML es la que marca la codificación. Qué es esto? Simplemente el formato en que se guardan los caracteres en el archivo. La codificación estándar es Unicode (utf-8)¹ y soporta caracteres de todos los idiomas: cirílico, castellano, chino, árabe, japones, coreano, ...) Debemos asegurarnos que nuestro editor de textos guarda el archivo en formato Unicode (opción a elegir en el cuadro de diálogo de Guardar como).

No lo tienes que aprender de memoria. Existen plugin² para los editores que lo agregan.

```
<meta charset="UTF-8">
```

Esa etiqueta es de <head>, por eso tiene esa forma. No volverás a usar ese tipo de etiquetas en tu documento.

Elije la opción que quieras, pero sólo una. Por cierto, si por motivos de espacio, la línea aparece cortada. No importa, el navegador interpreta los saltos de línea como espacios en blanco. En realidad, podrías escribir todo el archivo HTML en una sola línea. O cada palabra en una línea diferente. Muchos espacios en blanco seguidos se interpretan como uno.

Lo siguiente es indicar DOCTYPE, el cual le dice al navegador qué contiene el archivo que está abriendo: <!DOCTYPE html>

Después tienes a la cabecera y al cuerpo del programa entre las etiquetas <html> y </html>.

¹ Es un estándar de codificación de caracteres diseñado para el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas, además de textos clásicos de lenguas muertas. El término proviene de los tres objetivos: universalidad, uniformidad y unicidad

² Es una aplicación que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software. Puede nombrarse al plugin como un complemento.

El resto del documento tiene que ir dentro de la etiqueta `<html>`. Ella indica una serie de cosas, como qué idioma estamos usando. Si escribimos en castellano, quedaría así: `<html lang="es">`

Las abreviaturas son el código del idioma castellano. Veamos la siguiente tabla.

Idioma	Código
Castellano	es
Inglés	en
Francés	fr
Alemán	de
Japones	ja
Portugués	po
Euskera	eu

La cabecera (head) contiene información que no se ve directamente como parte del contenido de la página, como el título, vínculos a hojas de estilos CSS, información para robots de búsqueda, scripts, etc. Por ahora quedará sólo con el título de la página. La cabecera va encerrada entre `<head>` y `</head>`, mientras que para el título usa la etiqueta `<title>`. Éste, aparece en la barra de superior de la ventana del navegador, es el nombre que aparece si añade a favoritos la página, etc. Quedaría así:

```
1 <head>
2 <title>Titulo de la web</title>
3 </head>
```

El sangrado no es obligatorio, pero viene muy bien para leer el código.

Por último, tienes el cuerpo del programa entre `<body>` y `</body>`, y quedaría tal que así:

```
1 <body>
2
3 Aquí va el cuerpo del programa web
4
5 </body>
```

3.4.2 La plantilla

Recopilando todo, quedaría algo como esto:

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
5 <title>Titulo de la web</title>
6 </head>
7 <body>
8 <h1 align='center' >Viste la pelicula The Matrix?</h1>
9 </body>
10 </html>
```

código que debes grabar en un archivo llamado `plantilla.html` para su uso posterior.

3.4.3 Etiquetas básicas

Ahora que conocemos la estructura de un documento HTML, aprenderemos las etiquetas básicas para crear el contenido de su página web: párrafos, saltos de línea, títulos, etc.

3.4.3.1 Párrafos

Sirven para estructurar el contenido. En la web funcionan igual que en la realidad: contienen una o más frases relacionadas entre sí. Para crear un párrafo, escribimos texto entre las etiquetas `<p>` y `</p>`. Un ejemplo:

```
1 | <p>
2 |
3 | Hola , me llamo Luke Skywalker y soy piloto
4 |
5 | de una X-Wing en el Rogue Squadron. Tambien
6 |
7 | soy un Jedi del Lado Lumio de la Fuerza.
8 |
9 | Mis maestros han sido Yoda y Obi-Wan Kenobi.
10 |
11 | </p>
```

Da igual insertar un salto de línea entre la etiqueta y el contenido, ya que es interpretado como un espacio en blanco.

Si pruebas ese ejemplo, notarás que todo forma un párrafo y que el navegador pasa de los saltos de línea. Esto es correcto. Podríamos haber puesto todo el párrafo en la misma línea y obtener el mismo resultado.

3.4.3.2 Saltos de línea

Hay veces que necesita forzar un salto de línea dentro de un párrafo. Para ello, la etiqueta `
`, así:

```
1 | <p>
2 | DarkChestofWonders<br/>
3 | Seentroughttheeyes<br/>
4 | Oftheonewithpureheart<br/>
5 | Once,solongago.
6 | </p>
7 | Dark Chest of Wonders , de Nightwish , en el álbum Once.
```

Aunque estéticamente obtengamos el mismo resultado mediante párrafos (con `<p>`) que con saltos de línea de forma indiscriminada.

3.4.3.3 Los títulos (headings)

Los títulos sirven para agrupar información. Imagine la sección de enlaces de su página. El título principal podría ser "Mis links favoritos". Luego tendría los links clasificados por secciones, cada una de ellas bajo un subtítulo diferente: Blogs, Descargas y Videojuegos. Incluso podría tener subcategorías dentro de una misma sección, como por ejemplo RPG's, Aventuras Gráficas y Arcades.

Para esto, tienes las etiquetas `<hx>` y `</hx>`, donde x es un número del 1 al 6. `<h1>` corresponde al título más importante y sólo uno por página. Veamos:

```
1 | <h1>Mislinksfavoritos</h1>
2 | <h2>Blogs</h2>
3 | <!--blablabla-->
```

```

4 <h2>Videojuegos </h2>
5 <h3>RPG's</h3>
6 <!--blablabla-->
7 <h3>Arcades</h3>
8 <!--blablabla-->

```

Si vemos el código anterior en un navegador, aparecen los títulos de los importantes de mayor tamaño a los menos importantes (pero el tamaño de cada título siempre puede cambiarse con CSS).

3.4.3.4 Citas

Existen tres etiquetas para escribir citas: `<blockquote>`, `<q>` y `<cite>`. Mientras que `<blockquote>` y `<q>` se usan para escribir la cita en sí (las frases), `<cite>` se emplea para marcar el origen (persona, libro, canción o lo que sea).

Entonces, ¿Cuál es la diferencia entre `<blockquote>` y `<q>`? Pues que `<blockquote>` contiene citas largas. Es decir, que dentro de un `<blockquote>` pone párrafos. `<q>` funciona al revés, puesto que está pensada para escribir citas cortas. `<q>` va dentro de párrafos. La razón técnica es que `<blockquote>` es un elemento de bloque (block), mientras que `<q>` es inline y **NO** puede estar "aislado".

Vea un ejemplo de lo anterior:

```

1 <p>Aquí les dejo un fragmento de la canción
2 <cite>Die for Rock 'n' Roll</cite>, de Dover:</p>
3
4 <blockquote>
5 <p>
6 Everybody danced (while)<br/>
7 I was lying on the floor<br/>
8 I was ready to die<br/>
9 for Rock 'n' Roll<br/>
10 </p>
11 </blockquote>
12
13 <p>Mi parte preferida es cuando dice lo de
14 <q>I was ready to die[...]</q>.</p>

```

3.4.3.5 Separadores horizontales

Los separadores horizontales (horizontal rules) han caído en desuso, ya que conseguimos bordes delimitadores mediante estilos CSS. Pero como el saber no ocupa lugar. Veamos la etiqueta `<hr/>`:

```

1 <h2>Los videojuegos</h2>
2 <p>Bla bla bla...</p>
3 <hr/>
4 <h2>Musica</h2>
5 <p>Bla bla bla...</p>

```

3.4.3.6 Comentarios

Son notas que escribe en el código fuente de una página, pero no se muestran en pantalla. Para el navegador, son invisibles. Útiles para indicar qué hacen ciertas partes del código. Para insertar un comentario, lo escribe entre `<!-- y -->` y es una sola línea.

Ejemplo: `<!-- Esto es un comentario -->`

3.4.3.7 Enlaces

Ya conocemos las etiquetas necesarias para escribir texto. Ahora toca aprender a usar una de las características más importantes de la web: los enlaces (o links). Usamos la etiqueta `<a>`.

3.4.3.8 Enlace a una página externa

Si quieres enlazar a una página o archivo que está en otro servidor (normalmente webs que no son nuestras), `<a>` de esta forma:

```
<a href="http://www.algo.com" title="Descripcion">Texto del enlace</a>
```

El atributo href contiene la URL a la que quiere enlazar. Es importante que no olvide el protocolo (en este caso http://) o no funcionará.

En title escribe una descripción de la página que enlaza. Al igual que con `<acronym>` y `<abbr>`, en la mayoría de navegadores este título aparecerá al pasar el ratón por encima del link. No hay que confundir el título con el texto del enlace. Son completamente independientes.

Para enlazar a google.com:

```
<a href="http://www.google.com" title="La informacion que interesa">Google</a>
```

3.4.3.9 Enlace a una página local

Para enlazar a una página que esté en su servidor, necesita saber la ruta (path) desde el origen hasta la ubicación del archivo.

Si la página que contiene el link está en el mismo directorio que el destino (página a la que apunta el link), entonces sólo tiene que escribir su nombre:

```
<a href="profile.html" title="Mi Informacion">Ficha personal</a>
```

Si el destino está en un subdirectorio, utilice una barra / para indicar el camino:

```
<a href="galeria/color.html" title="Galeria color">Ver dibujos a color</a>
```

Si el destino está un directorio por encima, pondría dos puntitos y una barra ../ de esta manera:

```
<a href="../index.html" title="Pagina principal">Volver al inicio</a>
```

3.4.3.10 Enlace a una dirección de e-mail

Cree un enlace que, al presionar sobre él, se abra automáticamente una ventana del cliente de correo que tenga el usuario para que escriba un mensaje a esa dirección.

Para ello, sólo tiene que usar mailto: en el atributo href, seguido de una dirección de correo electrónico:

```
<a href="mailto:leia@alianza.net" title="E-mail de la Princesa Leia">Leia</a>
```

3.4.3.11 Anclas

Hemos aprendido enviar a las visitas a una página u otra de nuestra web.

Pero en ocasiones, tenemos páginas con contenido extenso que nos interesa enviar a las visitas no a una página sino a una área concreta de una página de nuestra web.

Usando las anclas o anclajes puedes hacer que cuando las visitas presione sobre el enlace, la ventana del navegador corra hacia esa área concreta de la misma página o pase a otra página distinta, pero no a su comienzo, sino a la parte media o a la parte que tú desees de esa otra página.

Para empezar hay que definir el área a la que quieres enviar a las visitas cuando hagan clic sobre ese enlace que pondremos después. Veamos cómo se define un ancla en la parte superior.

Si queremos enviar a las visitas a la parte superior, lo normal es colocar el enlace en la parte inferior (si el usuario está arriba, para que darle la opción de ir arriba si ya está allí). Esto se puede hacer con esta línea de Html:

```
<a href="#arriba" title="Ir Arriba">Ir arriba</a>
```

Si en una página de tu web escribes mucho texto y tienes 3 partes diferenciadas (3 títulos) podrías definir un ancla en cada uno de esos títulos, y luego colocar un menú en la parte inferior, o en la parte superior, o en ambas, dando la opción a las visitas a dirigirse, dentro de esa misma página, a la sección que deseen.

En ese caso colocarías al principio (o al final) ese menú, de este modo:

```
1 <a href="#titulouno" title="Ir al título uno">Ir al título uno</a>
2 <a href="#titulodos" title="Ir al título dos">Ir al título dos</a>
3 <a href="#titulotres" title="Ir al título tres">Ir al título tres</a>
```

Y ahora te faltaría colocar esas 3 anclas justo al lado de esos títulos, líneas de código como ésta:

```
1 <h1>Título Uno</h1><a name="titulouno"></a>
2 <p>Este es el texto correspondiente al título uno, aunque debería ser
  más largo para que se note el efecto.</p>
3 <h1>Título Dos</h1><a name="titulodos"></a>
4 <p>Este es el texto correspondiente al título dos, aunque debería ser
  más largo para que se note el efecto.</p>
5 <h1>Título Tres</h1><a name="titulotres"></a>
6 <p>Este es el texto correspondiente al título tres, aunque debería
  ser más largo para que se note el efecto.</p>
```

O podrías escribir:

`Leer comentarios` También, enlaza a un ancla que esté en otra página:

`Comentarios del post 5` Hay otro modo de crear anclas, y es usando el atributo "name" de una etiqueta a sin atributo href y sin texto para el enlace; pero esta manera es obsoleta. Tienen este aspecto (y son invisibles!):

```
<a name="comentarios" />
```

Y ya está todo lo que hay que saber sobre anclas. Son útiles para páginas muy largas, como las

FAQ³: tienen un índice de preguntas que enlazan al ancla de la respuesta correspondiente. Así, si hace clic en la pregunta número 3, enlazará a ese punto concreto de la página. Normalmente las respuestas tienen otro enlace que devuelve a un ancla situada en el índice de preguntas, para facilitar la navegación.

3.4.3.12 Listas

Ahora veremos cómo implementar listas en nuestras páginas. Las hay de tres tipos: ordenadas, sin ordenar, y de definición.

3.4.3.13 Listas ordenadas

Estas listas muestran sus elementos numerados. Se crean con la etiqueta ``:

```
1 <p> Mis escritores favoritos ( en orden de preferencia ) : </ p >
2 < ol >
3 < li >R . A . Salvatore </ li >
4 < li > George R R Martin </ li >
5 < li > Isabel Allende </ li >
6 </ ol >
```

Hay que tener en cuenta que con CSS cambia el número a empezar a contar, así como el tipo de numeración (números arábigos, roma, letras del abecedario, mayúsculas).

3.4.3.14 Listas sin ordenar

Las listas sin ordenar marcan cada elemento con una viñeta, de modo que no sigue una numeración. Se crean con la etiqueta ``.

```
1 <p>El helado perfecto: </p>
2 <ul>
3 <li>1 bola de helado de chocolate </li>
4 <li>1 bola de helado de limon </li>
5 <li>Trocitos de fresa y melocoton en almibar </li>
6 <li>Sirope de chocolate</li>
7 </ul>
```

Al igual que con las listas ordenadas, modifique su apariencia con CSS y elegimos el tipo de viñetas que quiere.

3.4.3.15 Listas de definición

Las listas de definición se diferencian de las anteriores en que cada ítem está compuesto por un par de elementos: un término y su definición. Se usan estas etiquetas: `<dl>` para crear la lista, `<dt>` para cada término y `<dd>` para las definiciones.

```
1 <p>Significado de algunos smileys : </p>
2 <dl>
3 <dt> : \ ) </dt>
```

³Preguntas y respuestas frecuentes

```
4 <dd>Sonrisa</dd>
5 <dt>xD</dt>
6 <dd>Carcajada</dd>
7 <dt>:P</dt>
8 <dd>Sacar la lengua</dd>
9 </dl>
```

Como comentario: las palabras término y definición no sólo se refieren a una palabra y a su significado. También, se usan una lista de definición para crear un perfil, relacionando los pares Nombre-Leia, Ciudad-Coruscant y Profesión-Senadora.

3.4.3.16 Listas anidadas

Las listas anidadas no son un nuevo tipo de lista, son una combinación de las anteriores. Anidar significa meter una lista dentro de otra. Veamos:

```
1 <p>Algunos libros de Salvatore :</p>
2 <ul>
3 <li>I Trilogía de El Elfo Oscuro
4 <ol>
5 <li>La Morada</li>
6 <li>El Exilio</li>
7 <li>El Refugio</li>
8 </ol>
9 </li>
10 <li>Trilogía de El Valle del Viento Helado
11 <ol>
12 <li>La Piedra de Cristal</li>
13 <li>Ríos de Plata</li>
14 <li>La Gema del Halfling</li>
15 </ol>
16 </li>
17 </ul>
```

Sólo debemos tener cuidado cerrando la etiqueta que toque. Cómo lo sabe? Fácil: primero se cierran las interiores, y después las de fuera. Es por esto, que cuando inserta una lista dentro de un ítem de otra lista (esto es, primero y después , luego cerrar la lista, y por último el ítem de la lista "exterior". Un buen sangrado como el del ejemplo ayuda mucho.

3.4.3.17 Imágenes

Las imágenes son un elemento importante para una web más atractiva, o de aportar más información. Conviene saber cuándo utilizarlas y no abusar de ellas.

Puede usar tres formatos de imagen: GIF, JPEG y PNG. El JPEG es adecuado para imágenes con muchos colores, como fotografías, y que no tengan grandes áreas de colores planos. El GIF sólo almacena hasta 256 colores diferentes, pero uno de ellos puede ser transparente. El PNG es el estándar y tiene varias profundidades de paleta (número de colores). Además, puede añadir un canal alpha para crear efectos con transparencias de diferente opacidad.

3.4.3.18 Insertar una imagen

Para colocar una imagen, usa la etiqueta `` con unos cuantos atributos:

```
<img src='image.gif' width='ancho' height='alto' alt='descripcion' >
```

Con `src` establece qué imagen mostrar. Al igual que con los links, hay que tener en cuenta la ruta de la imagen. Por organización, tenga las imágenes en un subdirectorio (o varios) llamado `imagenes`, así que escribimos `src='imagenes/algo.gif'`.

Los atributos `width` y `height` establecen ancho y alto de la imagen. Con un valor absoluto en píxeles o relativo en porcentaje. Así `width = "200"` muestra la imagen con 200 píxeles de ancho, mientras que `width = "100 %"` la imagen ocupa todo el ancho de la pantalla. Estos dos atributos no son obligatorios, pero es conveniente que indique las dimensiones en píxeles reales, ya que ahorra tiempo al navegador a la hora de maquetar la página.

El atributo `alt` contiene una descripción de la imagen, que se muestra cuando no se haya podido cargar por cualquier motivo. La mayoría de navegadores muestran esta descripción al pasar el ratón por encima de la imagen. Este atributo es obligatorio, por cuestiones de accesibilidad: hay personas que deshabilitan las imágenes para ahorrar tiempo; otras, usan navegadores de texto como Lynx.

Una cita muy común entre desarrolladores web suele ser: **El visitante más importante de tu web es ciego y se llama Google.**

Así podrías insertar la imagen de un banner:

```

```

3.4.3.19 Imágenes como links

También puede hacer que una imagen sea a su vez un enlace a una página. Los navegadores suelen mostrarla con un reborde para indicar que se trata de un link, pero lo cambia con CSS.

Para colocar una imagen como link, la introduce dentro de la etiqueta `<a>`:

```
1 <a href="http://art.jodocha.net" title ="Mi portafolio">
2   
3 </a>
```

Hay una técnica popular a la hora de implementar galerías de imágenes que consiste en usar thumbnails. Qué es esto? es una imagen pequeña de la original, de modo que al hacer clic sobre ella, carga la imagen a tamaño completo.

Entonces, algunos programadores hacen:

```
1 <a href ="matrix.jpg" title ="Wallpaper">
2   
3 </a>
```

Eso está mal. Si su protector de pantalla de Trinity ocupa 100KB (o más), los tiene como thumbnail. Es decir, justo lo que quiere evitar. El escalar una imagen con `width` y `height` no hace que pese menos. Mediante un programa de dibujo, escale la imagen, y guarde esta copia pequeña (de 5KB, por ejemplo).

3.4.3.20 Sobre el uso y abuso de imágenes

Dicen que una imagen vale más que mil palabras. Cierto, pero muchas imágenes, o pocas mal puestas, desesperan !!!

Le resulta esto familiar? Entra a una web con fondo de color chillón, letras verde brillante, una cantidad obscena de GIF animados, marquesinas, applets de Java, etc. Cuánto tarda en cerrar la página? menos de un segundo.

Es por esto que debe limitar los GIF animados, así como evitar el uso indiscriminado de imágenes. Recuerda: sólo hay que poner las imágenes necesarias. Además, ahorra en ancho de banda de tu servidor.

3.4.3.21 Tablas

Las tablas son el mecanismo que proporciona HTML para presentar información tabulada, como horarios o los resultados del fútbol.

Una tabla básica

Las principales etiquetas para crear una tabla son estas:

table	crear tabla
tr	crea fila
td	crea celda
th	crea título de celda
caption	crea título para la tabla

Tabla 2: Algunas etiquetas para tablas.

```
1 <table>
2 crea la tabla
3 <caption>
4 pone título a la tabla
5 <tr>
6 crea una fila
7 <td>
8 crea una celda
9 <th>
```

3.4.3.22 Atributos de table

La etiqueta <table> tiene de una serie de atributos que modifican borde y márgenes de las celdas.

Para cambiar el grosor del borde de la tabla, asigna a border con un valor en píxeles. Si no indicas nada, los navegadores toman como valor por defecto 1 o 0. Si no quieres borde, utiliza border="0".

Si lo que quieres es modificar la distancia entre una celda y otra, usa el atributo cellpadding con un valor en píxeles. Para modificar la distancia del contenido de la celda a sus bordes, utiliza cellspacing. La diferencia entre cellpadding y cellspacing puede confundirnos al principio, así que lo mejor es verlo con un ejemplo, modificando la tabla anterior:


```

1 <table border="1" cellpadding="10"
2   cellspacing="30">
3   <caption>Videojuegos </caption>
4   <tr>
5     <th>Título</th>
6     <th>Género</th>
7   </tr>
8   <tr>
9     <td>Sonic</td>
10    <td>Plataformas</td>
11  </tr>
12 </table>

```

3.4.4 Tablas para layouts?

Hoy, la mayoría de las páginas web están maquetadas usando tablas con `border="0"`. Antes de la llegada del CSS, era imposible crear texto a columnas y, en definitiva, maquetar un sitio web⁴. CSS implementa capas, con lo que configura totalmente la apariencia y la colocación de cada elemento de la página mediante la hoja de estilos, quedando el código HTML simple y sencillo.

Ahora aprenderemos a crear varios tipos de diseño populares.

El futuro es HTML y de todos los sitios web importantes es rediseñar su **layout** con capas. Te animo a que cumplas los estándares y no uses las tablas para maquetar.

3.5 CSS

En la parte de HTML ha comentado muchas veces que cambia el aspecto de una página web mediante CSS. Ahora es el momento de aprender cómo hacerlo. La mejor manera de aprender CSS es poniendo ejemplos de problemas comunes, junto con la manera de solucionarlo en CSS. Después de todo, este libro no pretende ser una guía de referencia con todas las propiedades y todos sus posibles valores.

3.5.1 Dónde escribo el código CSS?

Hay dos opciones para insertar CSS en un documento HTML. Lo más normal es hacerlo en un archivo externo (llamado "hoja de estilos") y enlazarlo desde su documento HTML. Esto tiene una ventaja, y es que puedes tener muchas páginas enlazando a la misma hoja de estilos. Si más adelante quieres cambiar algo, modifica un único archivo (la hoja de estilos) y afectarás a todas las páginas que lo usan. Enlaza una o más hojas poniendo esto dentro de la cabecera (`<head>`):

```
<link href="hoja\_estilos.css" rel="stylesheet" type="text/css" />
```

La otra opción es escribir la información referente a los estilos incrustada en el mismo archivo HTML. Lo insertas entre las etiquetas `<style>` y `</style>`, que también deben ir en la cabecera. Si hace las dos cosas a la vez, siempre tienen prioridad las reglas que estén dentro de `<style>`.

cómo funciona?

En una hoja de estilos utiliza reglas que consisten en elegir selectores a los que asignamos una serie de propiedades. Por ejemplo, si quieres que tu página web tenga el fondo blanco, las letras color gris, y una fuente Arial de tamaño 10 puntos, escribe:

```

1 | body {
2 |   background-color: #fff;
3 |   color: #666;
4 |   font-family: Arial, sans-serif;
5 |   font-size: 10pt;
6 | }

```

Las líneas terminan en un punto y coma. Es muy importante que no se te olvide. Donde #fff representa al color blanco, y #666 un color gris oscuro.

Selectores

Con ellos eliges a qué elementos se aplican las propiedades que escribe. Hay diferentes tipos de selectores, los más importantes son los verás ahora.

Si quieres elegir una etiqueta, simplemente escribe su nombre. Si quieres establecer las propiedades para los enlaces:

```

1 | a {
2 |   instrucciones
3 | }

```

Para elegir un elemento único utiliza su atributo id y la almohadilla:

```

1 | #menu {
2 |   instrucciones
3 | }

```

Otra cosa que puedes hacer es definir una clase y hacer que muchos elementos la utilicen, escribiendo un punto antes del nombre, veamos el ejemplo:

```

1 | .importante {
2 |   instrucciones
3 | }

```

Luego podríamos emplear esa clase en los párrafos que quiera (o cualquier otro elemento), usando el atributo class de este modo:

```

1 | <p class="importante">Bla bla bla</p>
2 | También, seleccione ciertos elementos, sólo cuando estén contenidos
   | dentro de otros. Si quiere seleccionar los <li>, pero sólo de las
   | listas sin ordenar:
3 | ul li {
4 |   ...
5 | }

```

Qué significa "cascading?" Cascading significa cascada y tiene que ver con la herencia. En CSS, los elementos hijos heredan todas las propiedades de sus padres. Si establece una regla para el elemento table, sus hijos (td entre otros) tendrán esa misma regla.

Es por esto que si quieres establecer un tipo de fuente para todo el documento, debes indicarlo en el elemento body para que se propague por toda tu página.

Puedo poner comentarios?

Claro, pero son distintos a los de HTML, que van entre <!-- y -->.

```
/* Esto es un comentario */
```

Por qué #fff significa blanco?

En CSS hay varias maneras de indicar un color. Puedes nombrarlo en inglés. En lugar de `#fff` escribes `white` y lo soluciona. El problema es incómodo porque tienes que aprender los nombres de cada color, y puede que exista algún color que quieras poner y que no tenga nombre. Además, de ser una falta de precisión absoluta. Así que normalmente se usa la notación en hexadecimal.

Los colores en su monitor están formados por tres haces de luz: rojo, verde y azul. Se llama sistema RGB (Red Green Blue). Mediante un programa de dibujo, elija un color indicando el valor de sus componentes rojo, verde y azul por separado.

1. Primarios aditivos saturados (absoluto)

- ROJO = `FF0000`
- VERDE = `00FF00`
- AZUL = `0000FF`
- Combinación de los tres primarios aditivos saturados (absolutos).
 - BLANCO = `FFFFFF`

2. Primarios sustractivos nulos (absolutos)

- CIAN = `00FFFF`
- MAGENTA = `FF00FF`
- AMARILLO = `FFFF00`
- Combinación de los tres primarios sustractivos nulos. (absolutos)
 - NEGRO = `000000`

3. Algunos secundarios y terciarios (media)

- GRIS = `808080`
- ROSA = `FF8080`
- VIOLETA = `800080`
- CELESTE = `80FFFF`
- MARRON = `800000`
- NARANJA = `FF8000`
- LAVANDA = `8000FF`
- TURQUESA = `00FF80`
- ORO VIEJO = `BBB200`

La mayoría de los colores que uses tendrán por cada componente los mismos dígitos. Por ejemplo: `#ff0000` (rojo), `#0000aa` (azul marino) o `#000000` (negro). Estos colores los puedes abreviar y dejarlos en tres cifras. Por ejemplo, `#f00` es completamente equivalente a `#ff0000`. Colores como `#a0a0a0` no son abreviados.

La mayoría de editores de código (X)HTML o de programas de dibujo muestran, en la paleta, el valor del color en hexadecimal. En cualquier caso, siempre puedes obtener el valor de sus componentes por separado y convertirlo a hexadecimal con una calculadora.

3.5.1.1 Colores, fondos y fuentes

Para empezar con CSS, debes conocer algunas propiedades sencillas.

El color de primer plano

La propiedad color hace referencia al foreground, es decir, al color que está por encima del fondo y viene a ser el color del texto. Si quiere que su página tenga las letras de gris oscuro, lo consigue con esto: `{ color : #666; }`

3.5.1.2 El fondo

Modifica el fondo de un elemento con la propiedad background, que tiene la siguiente sintaxis:

`background : color image repeat attachment position ;`

El primer parámetro corresponde al color de fondo, los siguientes son relativos a la imagen de fondo:

- image: indica la ruta a la imagen requerida. Por ejemplo, `url(fondo.gif)`.
- repeat: con esto establece si quiere que la imagen se repita o no, tanto horizontal como verticalmente. Con repeat se repite siempre en ambos sentidos (valor por defecto), mientras que con no-repeat no se repite nunca. Con repeat-x se repite sólo en horizontal, y con repeat-y sólo en vertical.
- attachment: indica si el fondo se queda fijo en el sitio o se desplaza con scroll. Con scroll (valor por defecto) el fondo se desplaza, y con fixed se queda siempre en el mismo sitio.
- position: indica la posición del fondo. Indicamos tanto la posición desde la izquierda como desde arriba (ya sea en píxeles, porcentajes, o incluso palabras).

En este caso, si puedes usar palabras, es preferible que lo hagas. Dispones de top (arriba), bottom (abajo), left (izquierda) y right (derecha).

Los valores por defecto son 0 % 0 %, que sitúan al fondo en la esquina superior izquierda. Si quiere las coordenadas 20,30 (tomando como el origen a la esquina superior izquierda), escriba 20px 30px. Si quiere el fondo centrado, 50 % 50 %.

Puedes omitir alguna propiedad si quieres. Además, establecer los valores de forma individual, con las propiedades background-color, background-repeat, etc.

Veamos algunos ejemplos para poner fondo a tu página:

```
1 | body {  
2 | background-color: #fff;  
3 | color: \#666;  
4 | font-family: Arial, sans-serif;  
5 | font-size: 10pt;  
6 | }
```

Hemos explicado de forma bastante detallada cómo funciona la propiedad background.

3.5.1.3 Fuente

Hay varias propiedades para jugar con el aspecto del texto. Puedes englobar todas bajo font, pero primero vea algunas de sus propiedades.

Considere que no todos los computadores tienen las mismas fuentes instaladas. Entonces qué hacer? Usar sólo fuentes "estándar", que tengan la mayoría de computadores. Puede especificar varias, de forma que si no se tiene la primera, se muestre la segunda, si no se tiene la segunda, pues la tercera, etc.

La propiedad que sirve para cambiar la fuente es `font-family`. Como acabamos de ver, puede indicar varias fuentes, por orden de preferencia. Si el nombre de una fuente tiene espacios en blanco, hay que ponerla entre comillas. Por ejemplo:

Haz esto y muere. La irrupción de la "Comic Sans" por doquier sólo es comparable a la aparición de "Jar Jar Binks".^{en} Star Wars.

```
font-family: 'Comic Sans MS', Arial, sans-serif;
```

EL tamaño de la fuente lo controla con `font-size`. Puedes indicar medidas en píxeles (px) o en puntos (pt). Puedes (y debes) usar medidas relativas, pero el tema de la herencia es muy importante en estos casos.

Para alinear el texto tiene a `text-align`, que toma los valores `right` (derecha), `left` (izquierda), `center` (centrado) o `justify` (justificado).

Es normal tener una clase así:

```
.centrar { text-align: center; }
```

Si quieres indentar los párrafos, pues use `text-indent`:

```
p { text-indent : 2em; }
```

Qué significa `em`? Es una unidad relativa. Puedes trabajar con píxeles o porcentajes, en algunos casos conviene utilizar otras unidades, 1 `em` es al tamaño de la fuente. Exactamente equivale a la anchura de la letra "M" mayúscula.

Así, si por herencia o por cualquier otra cosa el texto se muestra en un tamaño de fuente diferente, la proporción de indentado sería siempre la misma. Si pones 20px, siempre sería esa, aunque el tamaño de la fuente fuera de 20 o de 32 píxeles.

También cambia la decoración del texto mediante `text-decoration`. Puede tomar diversos valores, como `none` (sin adornos), `underline` (subrayado), `overline` (subrayado superior) o `line-through` (tachado). Si quieres que sus links no tengan subrayado, escribimos:

```
a { text-decoration : none; }
```

Podemos transformar las mayúsculas y minúsculas con la propiedad `text-transform`. Con `lowercase`, todo se mostrará en minúsculas; con `uppercase`, en mayúsculas; y con `capitalize` pones una letra capital al principio de cada palabra.

En cuanto al espaciado, para la distancia entre palabras usa `word-spacing`; para el de las letras, `letter-spacing`; y para el interlineado, `line-height`.

Un ejemplo con todo esto, para la etiqueta ``:

```
1 | strong {
2 |   color: #000;
3 |   letter-spacing: 0.25 em;
4 |   text-trasform: uppercase;
5 | }
```

Demasiada información? Sí, pero tenga en cuenta que no hay que memorizar todo. Lo importante es saber que existe una propiedad que hace tal cosa, no cómo se llama ni qué parámetros tiene. Para eso ya están las guías de referencia rápida y los editores de CSS.

3.5.1.4 El modelo de caja

Es hora de conocer uno de los fundamentos más importantes del CSS: el modelo de caja (box model). Es fácil, pero entenderlo bien es vital para realizar una buena maquetación de la web.

3.5.1.5 Cómo es el modelo de caja?

En realidad, todos los elementos de una web (párrafos, enlaces, imágenes, tablas, etc.) son cajas rectangulares. Los navegadores sitúan estas cajas de la forma que se les indique para maquetar la página. Hay dos tipos de cajas: block e inline. Los elementos block rompen el flujo de maquetación. Esto es, aparecen solos, insertando "saltos de línea". Los elementos inline siguen el flujo, y están contenidos dentro de elementos de bloque.

Por ejemplo, un párrafo sería un elemento block (no puede tener un párrafo al lado del otro, sino que dos párrafos seguidos aparecerán uno abajo del otro. En cambio, un enlace es un elemento inline, ya que no "corta" el texto donde está metido.

Las propiedades importantes de una caja son: width (ancho), height (alto), padding (relleno), border (borde) y margin (margen).

3.5.1.6 Ancho y alto

La propiedad width es un poco confusa, y durante mucho tiempo era horroroso trabajar con ella debido a que Ya-Sabes-Quién no la implementaba correctamente. Afortunadamente, desde la versión 6 del IE, width funciona como debería, así que es un quebradero de cabeza.

Lo que ha dado lugar a competiciones de a ver quién conseguía el chanchullo más bonito para sortear el bug. Si estás interesado, googlea buscando "box model hack".

La opción width representa el ancho de la caja. Pero es el ancho interior, es decir, si bordes, márgenes, ni padding. puede indicar este ancho en medidas absolutas (normalmente píxeles) o relativas (normalmente %). Aunque los elementos inline tienen width, si la modifica con CSS no verá resultado visual. Esto es porque el ancho de estos elementos se establece automáticamente para que se ajuste a las dimensiones del elemento inline. Si tiene un enlace que consiste en un texto de cinco caracteres, el ancho (width) de este elemento será lo que ocupen esos cinco caracteres.

Sobre el alto de la caja, se controla con la propiedad height, y todo lo que ha dicho antes sobre el ancho, también se aplica aquí.

3.5.1.7 Padding

Con padding estableces la distancia de "relleno" entre el límite interior de la caja y el exterior (borde). Si quiere poner un padding de 20 píxeles para toda la caja, lo hace así: `padding : 20 px;` Puedes establecer un padding distinto para cada lado, usando los sufijos -top (superior), -bottom (inferior), left (izquierda) y right (derecha):

```
1 <table border="1" cellpadding="10"
2   cellspacing="30">
3   <caption>Videojuegos </caption>
4   <tr>
5     <th>Título </th>
6     <th>Género </th>
7   </tr>
8   <tr>
```

```

9 | <td>Sonic</td>
10 | <td>Plataformas</td>
11 | </tr>
12 | </table>

```

Puedes abreviar lo anterior en una sola línea, indicando primero el padding superior y luego siguiendo el orden de las agujas del reloj. Es decir, queda: arriba, derecha, abajo, izquierda. El ejemplo anterior se acortaría así:

```
padding: 10px 20px 5px 30px;
```

Otro atajo útil es especificar sólo dos medidas: una corresponde al padding superior e inferior, y la otra al lateral. Si quiere que los paddings superior e inferior sean de 10 píxeles, y los laterales (izquierdo y derecho) de 20 píxeles, escribe:

```
padding: 10px 20px;
```

3.5.1.8 Bordes

Si quieres que tu caja tenga bordes, lo consigues con la propiedad `border`. Tiene la siguiente sintaxis: `border: width \ style \ color`

Como habrás supuesto, `width` especifica el grosor del borde. Normalmente es una medida en píxeles, pero también puede utilizar las palabras `thin` (fino), `medium` (normal) y `thick` (grueso). Por supuesto, qué tan ancho es `thick` queda a interpretación del navegador.

En cuanto a `style`, es el tipo de borde. Hay bastantes, pero los más comunes son: `solid` (línea continua), `dashed` (línea discontinua), `dotted` (línea de puntos) y `double` (línea continua doble).

Por último, `color` indica el color del borde.

Puede escoger un tipo de borde diferente para cada lado con los sufijos `-top`, `-bottom`, `-left` y `-right`. Para poner el borde inferior de 1 píxel a puntos rojos:

```
border-bottom: 1px dotted \#f00;
```

Para eliminar el borde, simplemente escribe que tiene de grosor 0 píxeles o que el estilo del borde es `none`.

Esto es importante con las imágenes, ya que si tiene una imagen enlazando a algo, los navegadores la ponen con un reborde feo. Así que esto se ha convertido ya en un fijo de las hojas de estilos:

```
img { border: none; }
```

3.5.1.9 Márgenes

Éstos se controlan con la propiedad `margin`, y es la distancia entre el borde de la caja y los elementos que la rodean.

En cuanto a la forma de usarla, es igual que con la propiedad `padding`, así que la forma de escribir y los atajos es exactamente la misma. Si quiere márgenes superior e inferior de 20 píxeles, y laterales de 5 píxeles:

```
margin: 20px 5px;
```

Para centrar un elemento de bloque, puedes hacer uso de `auto`:

```
margin: 0px auto;
```

3.5.1.10 Capas

Hablemos de una etiqueta HTML estrechamente ligada con CSS: `<div>`. Esta etiqueta crea una capa, que es un elemento de bloque que sirve de contenedor a otros elementos de bloque e inline.

Para qué sirve? Primero, para organizar semánticamente su página. El atributo `id` tiene carga semántica, así que si quiere poner en la cabecera de su web el título y el menú, haga esto: Cabecera en la maquetación. ¡No tiene nada que ver con la etiqueta `head`!

```
1 <div id="header">
2 <h1>Mi blog</h1>
3 <ul id="menu">
4 <li><a href="..." title="...">Principal</a></li>
5 <li><a href="..." title="...">Acerca de</a></li>
6 <li><a href="..." title="...">Enlaces</a></li>
7 </ul>
8 </div>
```

El otro uso de las capas es el de maquetar. Un layout típico de un blog tiene cuatro capas: la cabecera, el contenido principal, la barra lateral y el pie de página. Mediante CSS controla la disposición de estas capas, y conseguir el diseño que quiera.

3.5.1.11 Floats

Los floats son probablemente una de las cosas que más cuesta dominar. Los floats es alteran el flujo de la página, "incrustando" en él un elemento de bloque. El caso típico es colocar una imagen acompañando a un texto y que el texto "envuelva" la imagen. Esto lo consigue creando una clase como la siguiente:

```
1 .izquierda {
2     float: left;
3 }
```

Puedes indicar tanto `left` (izquierda) como `right` (derecha). Después de un float, pueden ocurrir sucesos extraños. La mayoría de ellos mediante la propiedad `clear` que "anula" los floats. Los valores que admiten son `left`, `right` y `both` ("ambos").

Volviendo a los layouts de blogs, es normal poner el contenido y la barra lateral con floats. Lo que pasa es que una de estas dos columnas pasa por encima del pie de página, en lugar de quedar el pie al final de todo. Esto se arregla así:

```
1 #footer {
2     clear: both;
3 }
```

3.5.2 Trucos con CSS

A continuación los ejemplos de CSS serán totalmente prácticos. Primero se muestran diferentes "trucos" para ciertas cosas. Pruebe y de comprenda todos ellos.

3.5.2.1 Links que cambian

Los enlaces tienen tres estados: sin visitar ("normales"), hover (pasar el cursor del ratón por encima), activos (hacer clic) y visitados (ha ido a esa dirección). Estos estados se corresponden

a pseudoelementos, y cambian la apariencia de los enlaces con CSS. Sin embargo, algunos navegadores tienen un bug que, dependiendo del orden en el que escriba las reglas, se mostrará el resultado correctamente o no. Cómo lo soluciona? Con un mnemotécnico que ayudará a recordar el orden en el que estas reglas funcionan bien en cualquier navegador.

Quedaría así:

```
1  /* links normales */
2  a:link {
3    text-decoration: none;
4    color: #00a;
5  }
6  /* visitados */
7  a:visited {
8    color: #a00;
9  }
10 /* hover */
11 a:hover {
12   text-decoration: underline;
13 }
14 /* activos */
15 a:active {
16   font-weight: bold;
17 }
```

El código anterior pone los enlaces de color azul marino y sin subrayado. Cuando pasamos el cursor por encima, aparece el subrayado. Al hacer clic, el texto del enlace estará negrita. Los enlaces que hayas visitados en color granate.

3.5.2.2 Links con el subrayado de diferente color

Al usar la propiedad text-decoration, el color de la línea de subrayado es el mismo que el del texto del enlace. Puedes hacer un pequeño truco para cambiar esto, y es quitar el subrayado y poner en su lugar un borde:

```
1  a {
2    color: #fff;
3    text-decoration: none;
4    border-bottom: 1px solid #f0c;
5  }
6
7  a:hover {
8    border: none;
9  }
```

Esto haría que los enlaces fueran de color blanco y que la línea de subrayado fuese fucsia. Al pasar el cursor por encima, se eliminaría el subrayado. El pseudoelemento hover funciona para cualquier otro elemento de la web. No dude en usarlo en celdas de tablas o capas.

3.5.3 Campos de formulario

Un efecto muy Web 2.0 es hacer los campos de formulario con fuente gigante y que el fondo del campo cambie cuando el usuario está usándolo. Esto se consigue con el pseudoelemento focus:

```
1 input, textarea {
2     font-size: large;
3     border: 3px solid #70C332;
4     color: #666;
5     background: #fff;
6 }
7
8 input:focus, text-area: focus {
9     background: #eee;
10 }
```

Lo anterior pondría los campos de formulario input y textarea con una fuente grande y gris oscura, borde ancho de color verde y el fondo blanco. Al hacer foco (el visitante está situado en ese campo), el fondo cambia a un color gris claro.

3.5.3.1 Blockquotes 2.0

Otro legado de la Web 2.0: blockquotes con texto gigante y unas comillas gigantes. Quizás es algo difícil de imaginar con esta descripción, pero si echas un vistazo a la figura 1, seguro que ya lo has visto en algún sitio antes.

Un ejemplo de código podría ser este:

```
1 blockquote {
2     background: url (comillas.gif) top left;
3     background-repeat: no-repeat;
4     text-indent: 30 px;
5     text-align: left;
6     font-size: x- large;
7     padding: 0px;
8 }
```

Tienes que ajustar los valores según la fuente y la imagen en sí. En muchos casos, es más conveniente utilizar padding y quitar el indentado.

3.5.3.2 Cambiar texto por imagen

Muchas veces, especialmente en headings, quiere cambiar el texto y en su lugar que aparezca una imagen. Un ejemplo del código sería este (suponiendo que su imagen mide 300x100 píxeles):

```
1 <!--HTML-->
2 <h3 id="enlaces"><span>Enlaces</span></h3>
3
4 /* CSS */
5 h3 # enlaces {
6     width: 300 px;
7     height: 100 px;
8     padding: 0px;
9     background: url(links.png) top left no-repeat;
10 }
```

```

11 | h3 span {
12 |   visibility: hidden;
13 | }

```

La etiqueta span es nueva, y no significa nada (literalmente). Es una etiqueta "vacía", y la usa para conseguir ciertos efectos CSS. Como semánticamente no tiene ningún valor, evitamos su uso en lo posible. Lo que hace el código anterior es hacer coincidir las dimensiones de `<h1>` con las de la imagen. Después, gracias al `` encargamos de que el texto del heading sea invisible.

No obstante, aunque el texto sea invisible, todavía existe. Por eso, si el título es muy largo, es posible que sea más grande que la imagen que usa de fondo, y es posible que descoloque el layout. En este caso, puede cambiar la fuente de `<h1>` y hacerla pequeña.

3.5.3.3 Crear un layout con CSS

Vamos a crear un layout sin tablas a dos columnas, típico en los blogs. Es un ejemplo sencillo de seguir.

Esta maquetación tiene las siguientes propiedades:

- Anchura fija.
- Centrado.
- 2 columnas (una de ellas de barra lateral).
- Cabecera (header)
- Pie de página (footer)

El esquema del código HTML de su página es (dentro del body):

```

1 | <div id="container">
2 |   <div id="header">
3 |     <h1>Título</h1>
4 |     <h2>Subtítulo</h2>
5 |   </div>
6 |   <div id="sidebar">
7 |     <h3>Sección</h3>
8 |     <p>Bla bla ...</p>
9 |   </div>
10 |  <div id="main">
11 |    <h3>Sección</h3>
12 |    <p>Contenido principal</p>
13 |  </div>
14 |  <div id="footer">
15 |    <p>Pie de página</p>
16 |  </div>
17 | </div>

```

Tiene cuatro capas. Veamos el código CSS de cada una de ellas. **#container** Esta capa es un contenedor para el resto de la página. Establece la amplitud de todo y centra el contenido. El truco está en usar `auto` dentro de `margin` para lograr el centrado. Esto no funciona en el IE, así que tiene que echar mano de `text-align: center` en el body. El CSS completo es este:

```

1 | body {
2 |     text-align: center;
3 | }
4 |
5 | #container {
6 |     width: 700px;
7 |     margin: 0px auto;
8 |     text-align: left;
9 | }
10 |
11 | #sidebar

```

Esta es la barra lateral en los blogs. En otras páginas web, puede poner menús, publicaciones, o cualquier otra cosa. La clave en esta capa es usar `float: left`, que la sitúa a la izquierda y hace que todos los demás elementos la rodeen. Código:

```

1 | #sidebar {
2 |     width: 200px;
3 |     padding: 10px;
4 |     float: left;
5 | }

```

El padding no es obligatorio, pero lo he puesto para después explicar la siguiente capa. También es necesario especificar el ancho en píxeles. Con este método la barra no se extenderá hasta abajo, sino que se corta en su final. Si pones la barra de otro color, y quieres que llegue hasta abajo, puedes usar la técnica de Faux Columns.

3.5.3.4 #main

Esta es la capa donde irá el contenido (en un blog, como los posts). Lo importante en esta capa es indicar con `margin-left` la distancia desde el borde del `#container` hasta esta capa, pasando por encima de la barra lateral.

Tu barra lateral tiene 200 píxeles de ancho y 10 píxeles de padding a la izquierda y a la derecha. Si haces memoria del modelo de caja, el margen izquierdo tiene que indicar $200 + 10 + 10 + X$, donde X es la cantidad que otros quiere dejar de separación entre la barra lateral y el contenido principal. Pondremos 5 píxeles:

```

1 | #main {
2 |     margin-left: 225px;
3 | }

```

#footer La capa del pie de página es para colocar información de copyright (©) o copyleft (Ⓒ) o cualquier otro tipo de datos misceláneos:

```

1 | #footer {
2 |     clear: both;
3 | }

```

¡tiene sus propio layout tablas que cumple los estándares del W3C!

3.5.3.5 Cabecera

Una de las cosas útiles que se han puesto de moda, es hacer que la cabecera de una web sea a su vez un enlace hacia la sección principal de una web.

Los estándares dicen que el título principal de la página se pone con `<h1>`. No obstante, el texto de esta etiqueta lo cambiaremos por una imagen. Además, quiere hacer un link, también usará `<a>`. Dentro del link, ponga una etiqueta `` para ocultar con CSS el texto y que sólo se vea la imagen que hará de cabecera. El código HTML es este:

```
1 |<!DOCTYPE html>
2 |<html lang='es'>
3 |<body>
4 |<div id=" header ">
5 |<h1><a href ="... " title ="...">
6 |<span>Título</span></a></h1>
7 |</div>
8 |</body>
9 |</html>
```

3.5.3.6 #header

Primero el CSS de la capa `#header`. Establezca el ancho y el alto de la capa con las medidas de la imagen (400x100 en este caso). Además, ponga la imagen de fondo y quite el margin y el padding:

```
1 |#header {
2 |    background: url(bg.png) top left no-repeat;
3 |    width: 400px;
4 |    height: 100px;
5 |    margin: 0px;
6 |    padding: 0px;
7 |}
```

3.5.4 heading

También quite los márgenes a la etiqueta `<h1>` (para que la capa mida exactamente lo que la imagen) y haga invisible el contenido de la etiqueta ``, para que el texto de la cabecera sólo se muestre en navegadores de texto o aurales:

```
1 |#header h1 {
2 |    margin: 0px;
3 |}
4 |
5 |#header a span {
6 |    visibility: hidden;
7 |}
```

3.5.5 Enlace

Ahora sólo falta la etiqueta `<a>`. Lo que quiere conseguir es que se pueda hacer clic en toda el área de la cabecera, no sólo en lo que sería el texto (invisible) del enlace. Para eso, tiene que

transformar el enlace en un elemento de bloque, y darle las medidas de la imagen.

Además, quitaremos los márgenes y el padding, así como la decoración del texto, para que no salga una línea de subrayado:

```
1 #header a {
2     width: 400px;
3     height: 100px;
4     display: block;
5     padding: 0px;
6     margin: 0px;
7     text-decoration: none;
8 }
```

Ahora tiene su cabecera con enlace incorporado.

3.5.6 Listas personalizadas

Cómo modificar las listas desordenadas para que tengan viñetas personalizadas. Para ello, necesita una imagen pequeña que haga de viñeta. En Internet hay muchas para descargar, aunque puede dibujar las tuyas propias. Suponga que la imagen se llama `bullet.png`.

Se trata de una lista sin ordenar:

```
1 <!DOCTYPE html>
2 <html lang='es'>
3 <body>
4 <ul>
5 <li>Sonata Arctica</li>
6 <li>Nightwish</li>
7 <li>HIM</li>
8 </ul>
9 </body>
10 </html>
```

3.5.7 Lista

Ahora la parte de CSS. Tiene una propiedad llamada `list-style-image`, pero con problemas con ciertas medidas de viñetas. Así que tiene que trabajar un poco el código. Para el elemento `ul` es:

```
1 ul {
2     padding-left: 10px;
3     margin-left: 10px;
4     list-style-type: none;
5 }
```

Lo importante es el `list-style-type: none`, que se encarga de quitar esas viñetas feas que pone el navegador por defecto. El padding y el margin es para sangrar la lista (puede poner los valores que quieras, o incluso quitarlos).

3.5.8 Los ítem

¡Ya sólo queda poner tu viñeta! Para ello, modifica el estilo de los `li` que estén en listas desordenadas:

```

1 | ul li {
2 |     padding-left: 12px;
3 |     background: url(bullet.png) 0em 0.5em no-repeat;
4 |     margin-bottom: 1em;
5 | }

```

Lo primero que verá es el padding por la izquierda. Esto no es el sangrado, es una distancia de relleno para que el texto del ítem no esté encima de la viñeta. Este valor lo modifica dependiendo de las dimensiones de la imagen de tu viñeta.

Luego, tiene la propiedad background. Después de establecer la imagen, debe indicar en qué posición (recuerde: primero horizontal, luego vertical) se encuentra la viñeta. Como ve, trabaja con em y no con medidas absolutas en píxeles. Por qué? Porque así vale para cualquier tamaño del texto.

Recuerde que 1em equivale a la anchura de la letra M.

Lógicamente, es muy difícil dar con el valor correcto a la primera, así que habrá que probar varias veces hasta encontrar el que mejor quede. No se puede olvidar el i para evitar el mosaico.

Por último, margin-bottom se encarga de establecer la separación entre un ítem y otro de la lista.

3.5.9 Menús verticales

Sabe que con listas se pueden hacer menús verticales? Por qué listas? Porque un menú vertical es una serie de elementos relacionados, y lo más semántico que puede hacer es meterlo en una lista.

El principal problema es que las listas son feas. Previamente, aprendió cómo hacerlas más bonitas, pero quizás quiera algo diferente para su menú. CSS da la solución, así que ya no tendrá excusa para hacer un menú en Flash. Qué es de lo peor que podría ocurrir? Si alguien no tiene instalado el plugin de Flash (nunca de por supuesto que el visitante tiene instalado un plugin), no podrá navegar por tu página.

En el menú que hará ahora, manejará los colores de fondo y los bordes para conseguir efectos cuando el ratón pase por encima. Puedes ver cómo quedaría en la figura 1. Como va con CSS, podría fácilmente incorporar imágenes, pero eso queda como ejercicio.

Necesita una lista así:

```

1 | <!DOCTYPE html>
2 | <html lang='es'>
3 | <body>
4 | <div id="menu">
5 | <ul>
6 | <li><a href="..." title="...">Home</a></li>
7 | <li><a href="..." title="...">Archivos</a></li>
8 | <li><a href="..." title="...">Enlaces</a></li>
9 | <li><a href="..." title="...">Acerca de</a></li>
10| </ul>
11| </div>
12| </body>
13| </html>

```

3.5.9.1 La lista

Empiece con dar la amplitud que quiera a la lista (que será la amplitud del menú), poner una fuente y quitar las viñetas y márgenes de la lista.

```
1 #menu ul {  
2     list-style-type: none;  
3     margin: 0px;  
4     padding: 0px;  
5     width: 200px;  
6     font-family: Arial, sans-serif;  
7     font-size: 11pt;  
8 }
```

A continuación, le dará color de fondo para los ítems de la lista (). Lo normal sería poner aquí los efectos de hover, para que se activen cuando el ratón pase por encima de todo el bloque, no sólo del texto del enlace. El CSS para el elemento li es:

```
1 #menu ul li {  
2     background-color: #666;  
3 }
```

3.5.9.2 Enlaces

Para hacer el hover: poner el enlace como si fuera un bloque, y así ocupará todo el li y podrá manipular sus dimensiones.

Haga memoria: es lo mismo que poner un enlace en la cabecera de la web.

```
1 #menu ul li a {  
2     color: #ccc;  
3     text-decoration: none;  
4     text-transform: uppercase;  
5     display: block;  
6     padding: 10px 10px 10px 20px;  
7 }
```

Sólo queda hacer los cambios para el hover:

```
1 #menu ul li a:hover {  
2     background: #000;  
3     border-left: 10px solid #333;  
4     color: #fff;  
5 }
```

Si lo pruebas, verás que las letras se desplazan al hacer el hover, debido a que aparece el borde izquierdo. Si no le gusta este efecto, puede añadir la línea siguiente al link cuando está normal. Lo que hace es poner un borde del mismo color que el fondo de los li, y así no existe:

`border-left: 10px solid #666;`

Merece la pena trabajar con esta técnica, dará lugar a menús muy logrados.

3.6 Entorno

Por qué se ve "biençon Internet Explorer y "malçon Firefox?

La realidad es que en IE se ve "mal", y en el resto de los navegadores (no sólo Firefox) se ve "bien". Pongo "bien" "mal" entre comillas, porque son apreciaciones que hace los humanos, subjetivas.

Los navegadores no son adivinos con bolas de cristal que se conectan a la mente del maquetador web e interpretan su voluntad. El maquetador tiene que dejar escrito, detalladamente, el contenido y apariencia de la página web: esto se consigue con los lenguajes HTML y CSS.

Estos lenguajes se encuentran bien definidos como estándar en el WWW Consortium (W3C). Este organismo se encarga de describir con precisión cómo deben interpretar el HTML y el CSS los navegadores.

Ahora bien, los navegadores no siempre cumplen lo que dice W3C.

Suponga que el Explorer "confunde" los colores rojo y blanco, y los intercambia, debido a un error de programación (no sabe si por descuido o deliberadamente). Es decir, que donde pone #fff IE lo interpreta como #f00, y viceversa. Suponga que el resto de navegadores interpretan los colores correctamente.

Qué pasaría si quisiéramos hacer una página web con fondo blanco? En su código CSS, pondríamos algo así:

```
body { background : #f00 };
```

Que en IE se mostraría blanco. Entonces cuando vamos a mirar la web con otros navegadores, vemos que se muestra de color rojo brillante. "En Firefox se ve mal". Pues no. Por mucho que se empeñe el Explorer en hacer creer a los desarrolladores web, el número #f00 significa rojo.

Lo que ha pasado es que una página se ha desarrollado mal (a menudo inconscientemente) para forzar a que se vea "bien" en IE. Lo que obtenemos es que en IE la página se visualiza incorrectamente, pero por casualidades místicas, esa visualización coincide con los deseos del diseñador.

Un caso real y muy gráfico de cómo IE visualiza mal las páginas lo puede encontrar en el Acid Test. Es un ejemplo de página web que construye mediante código estándar y válido un dibujo de una cara sonriente. Según lo bien programado que esté el navegador, veremos ese dibujo más o me bien.

3.7 Migración a HTML5

Si usabas HTML 4, puedes aprender muy rápido HTML5, puesto que las bases son las mismas. Sólo hay que tener en cuenta que HTML deja de lado todo aspecto estético y se centra en el contenido y en la semántica. En lugar de usar una etiqueta para dar un aspecto concreto, usa etiquetas para hacer que las palabras signifiquen una cosa u otra.

Veremos una serie de "reglas" a seguir para pasar a HTML. También es recomendable leer el capítulo donde se explica la estructura de un documento HTML. Evidentemente, no se muestran aquí todas las diferencias, pero sirve para hacer una idea de qué es lo que espera.

3.7.1 Minúsculas y comillas

Antes era una práctica común escribir las etiquetas en mayúscula, para diferenciarlas del código. Por compatibilidad con XML, en HTML todas las etiquetas deben ir en minúsculas.

Además, todos los atributos tienen que estar entre comillas dobles. Por ejemplo, si antes teníamos:

```
<IMG SRC=icono.gif ALT=Icono>
```

Ahora escribimos:

```

```

3.7.2 Cierre todas las etiquetas

Con HTML cree párrafos con la etiqueta `<p>` sin necesidad de cerrarla con `</p>`. Lo mismo ocurría con ``, por ejemplo. En HTML ninguna etiqueta puede quedar sin cerrar.

Lo de "ninguna.^{es} literal, hay etiquetas que no tienen como `<input>`, ``, `
` deben ser cerradas también. Cómo lo hace? Insertando la barra `/`, de forma que el salto de línea queda `
`. El espacio en blanco que hay entre el nombre y la barra es necesario para que los navegadores antiguos reconozcan la etiqueta.

**** He dicho que HTML deja de lado la apariencia del documento, ya que eso es controlado por CSS. Entonces, las etiquetas `` y `<basefont>` carecen de sentido.

Además, esos atributos de algunas etiquetas que hacen referencia al color de las cosas, imágenes de fondo, etc. también desaparecen por este motivo: son sustituidos por reglas CSS. Así que `background-color` a `bgcolor` y compañía. Lo mismo para el atributo `align` usado en párrafos e imágenes.

Ciertas etiquetas de formato, como `` (negrita), `<i>` (cursiva), etc. No se usan porque hacen referencia exclusivamente a la apariencia de las palabras. Si quiere dar énfasis, utilice ``, y para dar énfasis más fuerte ``. Los navegadores suelen mostrarlas como cursiva y negrita, respectivamente, aunque esto es lo de me porque puede cambiarlo con CSS.

3.7.3 Hay que usar alt y title

HTML hace hincapié en la accesibilidad de un documento, y por eso debe facilitar atributos de "apoyo.^a algunas etiquetas. La etiqueta `` dispone del atributo `alt`, que se muestra cuando la imagen no se puede cargar (a veces también cuando se pasa el cursor del ratón por encima). Hay que utilizarlo siempre.

Existe un atributo muy similar llamado `title`, que se utiliza en la etiqueta `<a>` (entre otras) y sirve para mostrar una descripción del sitio al que nos dirigimos. Por ejemplo:

```
<a href="http://www.google.es" title="Buscador">Google</a>
```

3.7.4 Cuidado al anidar etiquetas

HTML es muy estricto en cuanto a la anidación de etiquetas. Básicamente, hay dos tipos de elementos: los block y los inline. Los block son etiquetas como párrafos, headings, listas, etc. Los distinguimos porque siempre van solos e insertan saltos de línea. Los inline no interrumpen el flujo del texto. Son las etiquetas de formato, los enlaces, y demás. No puede meter un elemento block dentro de uno inline.

Sí quiere que el heading principal de su página sea un enlace, esto es incorrecto:

```
1 | <a href=".." title="..."><h1>Texto</h1></a>
2 | Hay que hacerlo así:
3 | <h1><a href=".." title="...">Texto</a></h1>
```

Además, hay ciertas etiquetas que no admiten otras dentro. Si tiene dudas, consulte el validador de código del WWW Consortium.

3.7.5 No existen los frames

Aunque la especificación HTML 1.0 Frameset permite el uso de frames (marcos) en una página, tanto la Transitional como la Strict los prohíben. Así que ya no puede usar ni frames ni inline frames.

De todos modos, aunque se pudieran emplear, no debería, ya que los frames son un atentado contra la usabilidad. Y a Google no le gustan, de paso.

No puede utilizar target !!!

Antes, la etiqueta <a> tenía un parámetro llamado target que permitía especificar en qué frame debía cargarse el destino de un link. Como ya no hay frames, este atributo es innecesario.

A los amantes del target="blank" para abrir webs en ventanas nuevas, si el visitante quiere abrir un vínculo nuevo, lo hará con el botón derecho del ratón. Y usar navegación con pestañas es molesto que se abran nuevas.

3.8 instancias del navegador

3.8.1 Las tablas no se usan para maquetar

Aunque puede crear un documento HTML con tablas para maquetar que pase el análisis del validador del W3C, va contra la filosofía de dejar HTML sólo para el contenido. Las tablas representar información tabulada, no para diseñar layouts.

De todos modos, pasar a un layout sin tablas no suele ser fácil, así que es conveniente familiarizarse con el HTML y CSS, y dejar la maquetación sin tablas para más tarde.

3.8.2 Los ampersands

Si tiene URL que contengan el ampersand (&), te llevarás la sorpresa de que el validador del W3C pelea con ellas. Esto es debido a las entities.

Qué son? Pues la manera que tiene de insertar caracteres de forma "segura". Por ejemplo, el carácter á se escribiría á. Como ve, las entities comienzan por un ampersand y acaban en punto y coma.

Por eso, cuando se leen las URL que contienen ampersands, hay confusión porque lo que sigue no es una entity. Así que tiene que sustituir el ampersand por su propia entity, &. Es decir, que esta URL:

```
http://alianza.net/p.php?nick=leia&show=all
```

Se queda en:

```
http://alianza.net/p.php?nick=leia&amp;show=all
```

Aunque, hablando de URL, también debemos emplear la entity & en nuestro texto normal y corriente cuando quiera escribir un ampersand.

3.8.3 Utiliza Unicode

A la hora de almacenar archivos de texto, puede elegir entre bastantes codificaciones, que viene a ser qué número (y de qué tamaño en bits) se asigna a cada carácter.

Qué es lo que pasa cuando abra un archivo en una codificación (como, la Shift-JIS que es una japonesa) y le dice que la lea con otra (la ISO 8850-1, de caracteres latinos)? Obtiene símbolos extraños. Esto pasa, cuando alguien envía un trackback con una codificación distinta a la que uso en el blog.

Los trackback son un tipo especial de "comentarios" en los blogs. Cuando alguien de otro blog te enlaza al tuyo, si te manda un trackback, aparecerá en tu post un comentario con la URL del blog de la otra persona. Es útil para saber quién habla sobre cosas que has dicho.

Pero, qué pasa si quieres hacer un documento multilingüe con distintos alfabetos? Qué pasa si desde una aplicación quiero leer un documento en cualquier idioma? Para esto existe una codificación de caracteres estándar, que es la Unicode.

Unicode proporciona un número único para cada carácter, sin importar la plataforma, ni el programa, ni el idioma.

Existen varios formatos de Unicode, el más usado es el UTF-8 y está bastante extendido en la Red. De hecho, en la blogosfera las dos codificaciones más usadas son la ISO 8859-1 y la UTF-8. Usar Unicode es bueno y es la codificación estándar.

3.9 Formularios

Los formularios recogen información introducida por el visitante de su web. Esta información puedes enviarla por correo electrónico o procesarla con un script.

Si envías un correo electrónico, es importante tener en cuenta de que se trata de información no cifrada y que podría ser interceptada, así que no debe contener datos importantes. Un uso aceptable sería un formulario con comentarios sobre su página o para pedir un intercambio de links.

Si su servidor dispone de tecnología como PHP o CGI, puede hacer más cosas con esa información, como guardarla en una base de datos o generar una página dinámica.

3.9.1 La etiqueta FORM

Todos los formularios están enmarcados por `<form>` y `</form>`. Dentro de esta etiqueta, van los campos del formulario, y puede usar párrafos y saltos de línea para organizarlos. Vea un ejemplo de etiqueta `<form>`, suponiendo que va a ser un formulario que se enviará por correo electrónico:

```
<form action="mailto:leia@alianza.net" method="post" enctype="text/plain">
```

El atributo `action` recoge qué hacer una vez que se presione el botón de enviar (Submit). En ese ejemplo, el formulario se envía a la dirección `leia@alianza.net`³. Si tuviésemos un script para procesar el formulario, podríamos algo como `action="enviar_info.php"`.

Con `method` especifica cómo va a ser enviada la información. Si utiliza correo electrónico, asigne el valor de `post`. Para scripts se utiliza `get`, `put`, `request` que pone el valor de las variables en la query string (URL).

Por último, con `enctype="text/plain"` consigue que llegue a su bandeja de entrada el formulario en

forma de texto plano sin caracteres extraños.

3.9.2 Campos de texto

La mayoría de los campos de un formulario se crea con una sola etiqueta, `<input>`, y mediante su parámetro `type` especificamos el tipo de campo que quiere. Un campo de texto básico queda así:

```
<input type="text" id="nombre" name="nombre" size="20" />
```

Veamos los atributos. Con `type="text"` indica que se trata de un campo de texto. El atributo `size` recoge el ancho del campo, medido en caracteres.

Ahora bien, qué hacen `id` y `name`?

Pues `id` es un identificador. Esto implica que nada en todo el documento puede llamarse igual a este elemento.

Este parámetro sirve para CSS y para más cosas, como usarlo con la etiqueta `<label>`.

Con `name` asigna nombre a la variable de ese campo. Si el visitante escribe "Morpheo" en el formulario que ha puesto de ejemplo, recibirá un e-mail que contendría algo así:

```
nombre = Morpheo
```

De todos modos, para ahorrar problemas, siempre que pueda es mejor escribir el mismo valor para `id` y `name`. Hay que complicarse la vida lo menos posible.

Hay otros atributos adicionales para nuestros campos de texto. Podemos indicar el número máximo de caracteres a introducir por el usuario con `maxlength` Y si quiere que aparezca un valor por defecto, use `value="algo"`. Para pedir la dirección de la página web del visitante:

```
<input type="text" name="url" id="url" size="30" maxlength="255" value="http ://"/>
```

Y, por supuesto, no olvidar su `title`, que funciona igual que con la etiqueta `<a>`.

3.9.3 Campos de contraseña

Los campos de contraseña son exactamente iguales que los de texto, sólo que el visitante en lugar de ver los caracteres introducidos, ve asteriscos. Lo de "exactamente iguales" quiere decir eso: es un campo de texto, la información no va cifrada de ninguna manera. La diferencia entre un campo de texto y uno de contraseña es meramente estética.

Los atributos son los mismos que los de los campos de texto, lo único que cambia es que debe introducir `type="password"`. Ejemplo:

```
<input type="password" name="pass" id="pass" />
```

3.9.4 Etiquetar campos

Ha aprendido a crear campos de texto para su formulario, pero cómo le decimos al visitante qué es lo que tiene que introducir en cada campo? podría hacer:

```
1 <p>Nombre :  
2 <br />  
3 <input type="text" name="nombre" id="nombre" />  
4 </p>
```

Mal. Puedes tener problemas con navegadores no visuales. Entonces, cómo sabemos que la palabra "Nombre" hace referencia al campo con el atributo `id="nombre"`? Para eso disponemos de una

etiqueta nueva: <label>.

Esta etiqueta se encarga de asociar texto con su campo correspondiente. Sólo tenemos un atributo, for, e indicamos la id del campo al que hacer referencia. El ejemplo anterior es correcto de esta manera:

```
1 <p>
2 <label for="nombre">
3 Nombre :
4 </label>
5 <br />
6 <input type="text" placeholder="escriba su nombre completo" name="
  nombre" id="nombre" />
7 </p>
```

3.9.5 Áreas de texto

Con las áreas de texto dará a sus visitantes la posibilidad de introducir texto con varias líneas. La etiqueta a usar es <textarea>, y su funcionamiento es diferente al de <input>.

La etiqueta <textarea> dispone de los atributos id, name y title, que funcionan como en el resto de campos de formulario. Además, dispone de otros dos para indicar las dimensiones del área de texto: cols se encarga de establecer el ancho (medido en caracteres) y rows el alto, medido en líneas.

Si en una parte del formulario quiere poner un campo para que el visitante deje un comentario, el cual probablemente sea largo, utilice un textarea:

```
1 <p>
2 <label for="comentario">
3 Algun comentario?
4 </label>
5 <br />
6 <textarea name="comentario" id="comentario" cols="30" rows="5">
7 Bla bla bla
8 </textarea>
9 </p>
```

Fíjate que <textarea> tiene etiqueta de cierre. El texto entre la etiqueta de apertura y la de cierre es el valor que contendrá el campo; en este caso "Bla bla bla".

3.9.6 Casillas de verificación

Una casilla de verificación (conocida como checkbox) es un cuadro que el usuario activa y desactiva presionando en él. Se crean con la etiqueta <input> y type="checkbox". Los atributos id, name y title funcionan con normalidad, pero value funciona de manera algo distinta. Lo que escriba en value es lo que saldrá en el e-mail que reciba como el valor de la variable (indicada en name) si la casilla está activada. Es decir, que si pones esto:

```
1 <p>
2 Has jugado a ...
3 <br />
4
5 <input type="checkbox" name="monkey1" id="monkey1" value="si" />
```

```

6 | <label for="monkey1">
7 |
8 | Monkey Island I
9 | </label>
10| </p>

```

Y el usuario activa la casilla, recibirá un mail como este:

```
monkey1 = si
```

También hace que una casilla esté activada por defecto, añadiendo el atributo checked="checked".

3.9.7 Botones de selección

No sé si el término "botones de selección" es el adecuado como traducción, así que mejor pondré una explicación de lo que hacen. El nombre en inglés es radio buttons, y son casillas circulares agrupadas, en las que sólo una puede ser activada a la vez. Sirven cuando quiere que el visitante sólo seleccione una opción de las múltiples que se le dan.

Aunque se crean con la etiqueta <input> indicando el parámetro type="radio", los radio buttons son algo "especiales", así que vaya despacio.

Ponga el caso de que quiere que el visitante de su web indique cuál película de la trilogía de "Star Wars" es su favorita. Evidentemente, sólo puede seleccionar una, así que a emplear radio buttons en vez de casillas de verificación. necesita un botón por cada película (tres en total). Cómo los agrupa? Pues dando el mismo nombre de variable a cada botón.

Es decir, el atributo name es el mismo para todo el grupo. Y qué hace con id? Bien, no puede haber dos valores de id repetidos, así que la id en cada botón ha de ser distinta. En otras palabras, los radio buttons son los únicos campos en los que la id y name deben ser diferentes. Dispone del atributo checked por si quiere marcar alguna opción por defecto. El código para este ejemplo sería:

```

1 | <p>
2 | Película preferida :
3 |
4 | <input type="radio" name="peli" id="sw_hope" value="hope" checked="
   | checked" />
5 | <label for="sw_hope">
6 | A New Hope
7 | </label>
8 | <input type="radio" name="peli" id="sw_empire" value="empire" />
9 | <label for="sw_empire">
10| The Empire Strikes Back
11| </label>
12| <input type="radio" name="peli" id="sw_jedi" value="jedi" />
13| <label for="sw_jedi">
14| The Return of the Jedi
15| </label>
16| </p>

```

Como ve, usa como nombre de variable (name) la palabra "peli". Según la película que sea, cada botón tiene asignada una id diferente. Así, "A New Hope" tiene asignada la id "peli_hope". El atributo value contiene el texto que tendrá la variable en caso de que se seleccione ese botón. Si el visitante selecciona la película de "The Return of the Jedi", recibirá un e-mail con esta línea:

pele = jedi

3.9.8 Listas de selección

Las listas de selección tienen una función parecida a los radio buttons, en tanto que se presentan múltiples opciones agrupadas en las que escoge una. La diferencia es que la lista aparece replegada y no ocupa apenas espacio en la web, así que son útiles cuando tiene muchas opciones a elegir.

La etiqueta `<select>` crea la lista de selección y `</select>` la cierra. Entre ellas, inserta las opciones que tiene con la etiqueta `<option>`. Pongamos el mismo ejemplo de antes, seleccionando la película preferida de "Star Wars".

```
1 <p>
2 <label for="pele">
3 Película preferida :
4 </label>
5
6 <select name="pele" id="pele">
7 <option value="hope">
8 A New Hope
9 </option>
10 <option value="empire">
11 The Empire Strikes Back
12 </option>
13 <option value="jedi" selected="selected">
14 The Return of the Jedi
15 </option>
16 </select>
17 </p>
```

Con `selected="selected"` indica cuál es la opción por defecto, en este caso **The Return of the Jedi**.

3.9.9 Botones de enviar y reestablecer

Vistos todos los campos de formulario que podemos crear, ahora sólo falta comentar dos botones especiales: el de enviar (submit) y el de reestablecer (reset).

Ambos se crean con `<input>`. El atributo `id` no tiene mucho sentido, a menos que use CSS para cambiar su aspecto de un modo concreto y exclusivo. Así mismo, `name` tampoco es útil si no algún tipo de script para tratar la información. Esto por el momento.

En `value` indica el texto que aparece en el botón. El botón de enviar se encarga de mandar la información del formulario, según lo que haya especificado en `<form>`. Para crear el botón, simplemente `type="submit"`:

```
<input type="submit" value="Enviar" />
```

El botón de reestablecer borra el formulario y vuelve a poner los valores por defecto. Útil para formularios largos. Lo consigue con `type="reset"`:

```
<input type="reset" value="Borrar" />
```

Tienes diferenciar bien cuál botón es cuál y no poner textos extraños como título de los botones.

3.10 Ejercicios

3.10.1 Práctica 1

La página web de "La chistera"

La propuesta

En esta primera práctica se propone crear un sitio Web REAL poniendo en práctica los conocimientos adquiridos hasta ahora. En él incluiremos imágenes, varios titulares, párrafos de texto de distinto tipo y enlaces hipertexto a otras páginas.

Visite y capture cualquier página web. Se propone al lector intentar imitar la apariencia de esta página usando únicamente las etiquetas HTML explicadas hasta ahora.

3.10.2 Práctica 2

Un sitio web completo

La propuesta

Ya puede dar por concluida la primera fase del aprendizaje del HTML. En poco tiempo ha aprendido como:

- Es la estructura general de una página: para crear esta estructura usa las etiquetas HTML, HEAD y BODY.
- Dar título a la página con la etiqueta TITLE.
- Crear encabezados con las etiquetas H1, H2, H3, H4, H5, H6.
- Introducir párrafos de texto normal con la etiqueta P y alinearlos a sus gusto.
- Crear otros párrafos con BLOCKQUOTE y ADDRESS, y en este mismo capítulo párrafos preformateados con PRE.
- Dividir el texto con saltos de línea o líneas horizontales de distintos grosores y tamaños.
- Insertar en nuestras páginas los dos elementos que más fama han dado al World .
- Es el Wide Web: los enlaces hipertexto y las imágenes.
- Dar distintos formatos al texto, tanto formatos lógicos (EM, STRONG, CODE, etc.) como físicos (B, I, TT, BIG, SMALL, SUB, SUP, etc.).
- Aplicar los diversos tipos de listas (UL, OL, DL, MENU y DIR).

Como vemos la lista es larga, pero ya estas equipado con conocimientos suficientes para crear una página compleja.

Para afianzar estas bases lo más importante es la práctica, y para no quedar sólo en las palabras vamos a empezar ya con la segunda práctica del curso. Ésta consistirá en la realización de un sitio web completo que, aunque es sencillo (dos páginas), permitirá al lector hacerse una idea de los procesos que hay que llevar a cabo en la elaboración de webs en el mundo profesional actual.

El sitio web que crearemos puede ser de una empresa de tu propiedad, o de una que ha contratado tus servicios. Hemos elegido como ejemplo una agencia de viajes, pero te invitamos a buscar otro

tema, ficticio o real, y crear tu propia página web, si así lo prefieres.

A lo largo de la creación de la página iremos detallando los pasos seguidos para que puedas repetirla o crear otra similar en tu computador.

Capítulo 4

Diseño de Base de datos

Esta unidad explica en qué consiste el diseño de una base de datos, analiza las etapas en las que se puede descomponer y describe brevemente las etapas del diseño conceptual y lógico de una base de datos relacional.

4.1 Objetivos

1. Conocer las etapas que integran el proceso del diseño de una base de datos.
2. Conocer las estructuras del modelo ER¹.
3. Saber hacer el diseño conceptual de los datos de un sistema de información mediante el modelo ER.
4. Saber hacer el diseño lógico de una base de datos relacional partiendo de un diseño conceptual expresado con el modelo ER.

4.2 Qué es una base de datos?

Una base de datos almacena datos que utiliza un sistema de información determinado. Debe tener en cuenta las necesidades y los requisitos de los usuarios del sistema de información para tomar adecuadas decisiones.

El diseño de una base de datos define la estructura de los datos que debe tener la base de datos en el sistema de información. En el caso relacional, esta estructura es un conjunto de esquemas de relación con sus atributos, dominios, claves primarias y foráneas, etc.

4.3 Etapas del diseño de bases de datos

1. Etapa del diseño conceptual: se obtiene una estructura de la información de la BD², independiente de la tecnología a emplear. Aún no considera qué tipo de base de datos utilizará –relacional, orientada a objetos, jerárquica, de red, etc.–; tampoco tiene en cuenta con qué

¹Entidad-Relación

²Base de datos

SGBD³ ni con qué lenguaje concreto se implementará la base de datos. El diseño conceptual trata únicamente con la estructuración de la información, sin resolver cuestiones tecnológicas.

El resultado del diseño conceptual se expresa mediante algún modelo de datos de alto nivel. Uno de los más empleados es el modelo entidad-relación o simplemente ER.

2. Etapa del diseño lógico: una vez obtenido el resultado del diseño conceptual, procede a su transformación de forma tal que se adapte a la tecnología a emplear. Concretamente, es preciso que se ajuste al modelo del SGBD con el que se desea implementar la base de datos.
3. Etapa del diseño físico: luego del diseño lógico, el resultado se transforma la estructura obtenida con el objetivo de conseguir una mayor eficiencia; además, se completa con aspectos de implementación física que dependerán del SGBD.

En la etapa del diseño físico –con el objetivo de conseguir un buen rendimiento de la base de datos–, debe tener en cuenta las características de los procesos que consultan y actualizan la base de datos, así como los caminos de acceso que utilizan y las frecuencias de ejecución. También, considere los volúmenes que espera tener de los diferentes datos que quiere almacenar.

4.4 Diseño conceptual: modelo Entidad-Relación

El modelo ER es uno de los enfoques de modelización de datos más utilizado por su simplicidad y legibilidad. Su legibilidad se favorece por proporcionar un diagrama comprensivo. Es una herramienta útil tanto para que el diseñador refleje en un modelo conceptual los requisitos del mundo real de interés como para comunicarse con el usuario final sobre el modelo conceptual obtenido y, de este modo, verificar si satisface sus requisitos.

El modelo ER resulta fácil de aprender y de utilizar en la mayoría de las BD. Además, existen herramientas informáticas de ayuda al diseño (herramientas CASE⁴) que utilizan alguna variante del modelo ER para hacer el diseño de los datos.

El nombre completo del modelo ER es entidad-relación, y proviene del hecho de que los principales elementos que incluye son las entidades y las relaciones.

El origen del modelo ER proviene de trabajos efectuados por Peter Chen, 1976.

Utilice el modelo ER para comunicarse con el usuario, es recomendable emplear una variante del modelo que incluya sólo sus elementos más simples –entidades, atributos y relaciones– y, tal vez, algunas construcciones adicionales, como entidades débiles y dependencias de existencia.

4.4.1 Entidades, atributos e relaciones

Una entidad es un objeto del mundo real que distinguimos del resto de objetos y del que nos interesan algunas propiedades.

4.4.1.1 Ejemplos de entidad

Como ejemplos de entidad tenemos empleado, producto o despacho. También son entidades otros elementos del mundo real de interés, menos tangibles pero igualmente diferenciados del

³Sistema gestor de base de datos

⁴Ingeniería de software asistida por computadora

resto de objetos; como una asignatura impartida en una universidad, un préstamo bancario, un pedido de un cliente, etc.

Las propiedades de los objetos que nos interesan se denominan atributos.

4.4.1.2 Ejemplos de atributo

Sobre una entidad empleado nos interesa registrar su CI (dni en la figura), su NSS, su nombre, su apellido y su sueldo como atributos. Veamos la figura 4.1:



Figura 4.1:

El término entidad se utiliza tanto para denominar objetos individuales como para hacer referencia a conjuntos de objetos similares de los que nos interesan los mismos atributos; es decir, que se utiliza para designar tanto a un empleado en concreto como al conjunto de todos los empleados de la empresa. El término entidad se puede referir a instancias u ocurrencias concretas (empleados concretos) o a tipos o clases de entidades (el conjunto de todos los empleados).

El modelo ER proporciona un diagrama para representar gráficamente las entidades y sus atributos:

- Las entidades se representan con un rectángulo. El nombre de la entidad se escribe en mayúsculas dentro del rectángulo.
- Los atributos se representan mediante su nombre en minúsculas unido con un guión al rectángulo de la entidad a la que pertenecen. Muchas veces, dado que hay muchos atributos para cada entidad, se listan todos por separado del diagrama para no complicarlo.

Cada uno de los atributos de una entidad toma valores de un cierto dominio o conjunto de valores. Los valores de los dominios deben ser atómicos; es decir, no deben ser descompuestos. Además, todos los atributos tienen que ser univaluados, es decir que tiene un único valor para cada ocurrencia de una entidad.

4.4.1.3 Ejemplo de atributo univaluado

El atributo sueldo de la entidad empleado, toma valores del dominio de los reales y únicamente toma un valor para cada empleado concreto; por lo tanto, ningún empleado puede tener más de

un valor para el sueldo.

Una entidad debe ser distinguible del resto de objetos del mundo real. Una entidad debe tener un atributo que lo hace único. Este atributo forma una clave de la entidad.

4.4.1.4 Ejemplo de clave

La entidad empleado tiene una clave, el atributo ci, porque todos los empleados tienen números de CI diferentes.

Una determinada entidad puede tener más de una clave; es decir, puede tener varias claves candidatas.

4.4.1.5 Ejemplo de clave candidata

La entidad empleado tiene dos claves candidatas, la que está formada por el atributo ci y la del atributo nss, teniendo en cuenta que el NSS también será diferente para cada uno de los empleados.

El diseñador elige una clave primaria entre todas las claves candidatas. En el diagrama, la clave primaria se subraya para distinguirla del resto de las claves.

4.4.1.6 Ejemplo de clave primaria

En el caso de la entidad empleado, elige ci como clave primaria.

Se define relación a la asociación entre entidades.

Las relaciones se representan en los diagramas del modelo ER mediante un rombo. Junto al rombo se indica el nombre de la relación con letras mayúsculas.

4.4.1.7 Ejemplo de relación

Considere una entidad empleado y una entidad despacho y suponga que a los empleados se les asignan despachos donde trabajar. Entonces hay una relación entre la entidad empleado y la entidad despacho.

Esta relación, que denomina asignación, asocia a los empleados con los despachos donde trabajan. La figura 4.2 muestra la relación asignación entre las entidades empleado y despacho.

El término relación se utiliza tanto para asociaciones concretas u ocurrencias de asociaciones como para conjuntos o clases de asociaciones similares.

4.4.1.8 Ejemplo

Una relación se aplica tanto a una asociación concreta entre el empleado de CI '50.455.234' y el despacho 'Diagonal, 20' como a la asociación genérica entre la entidad empleado y la entidad despacho. Ver figura 4.2.

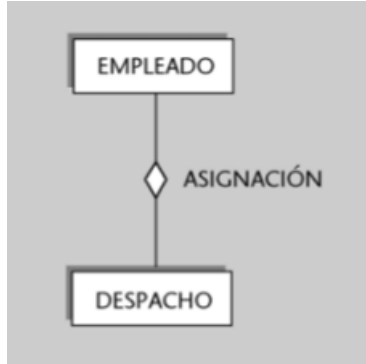


Figura 4.2:

4.4.2 Grado de las relaciones

Una relación puede asociar dos o más entidades. El número de entidades que asocia una relación es el grado de la relación. Veamos la figura 4.3.

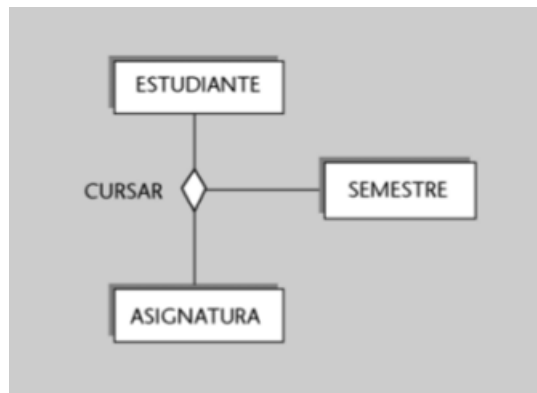


Figura 4.3:

La relación ternaria evaluación-semestral asocia estudiantes, asignaturas y una tercera entidad que denominamos semestre. Su atributo nota refleja todas las notas de una asignatura que tiene un estudiante correspondientes a diferentes semestres.

4.4.2.1 Relaciones binarias

La conectividad de una relación expresa el tipo de correspondencia que se establece entre las ocurrencias de entidades asociadas con la relación. En el caso de las relaciones binarias, es el número de ocurrencias de una de las entidades con las que una ocurrencia de la otra entidad puede estar asociada según la relación.

Una relación binaria entre dos entidades puede tener tres tipos de conectividad:

- Conectividad uno a uno (1:1). Se denota poniendo un 1 a lado y lado de la relación.
- Conectividad uno a muchos (1:N). Se denota poniendo un 1 en un lado de la relación y una N en el otro.

- Conectividad muchos a muchos: (M:N). Se denota poniendo una M en uno de los lados de la relación, y una N en el otro.

4.4.2.2 Ejemplos de conectividad en una relación binaria

Analizamos un ejemplo de cada una de las conectividades posibles para una relación binaria:

1. Conectividad 1:1



Figura 4.4:

La relación anterior tiene conectividad 1:1. Ésta, asocia los estudiantes de un semestre con las asignaturas cursadas. El hecho de que sea 1:1 indica que un estudiante tiene sólo un semestre, y que cada semestre está situado en una única asignatura. Ver la figura 4.4.

2. Conectividad 1:N

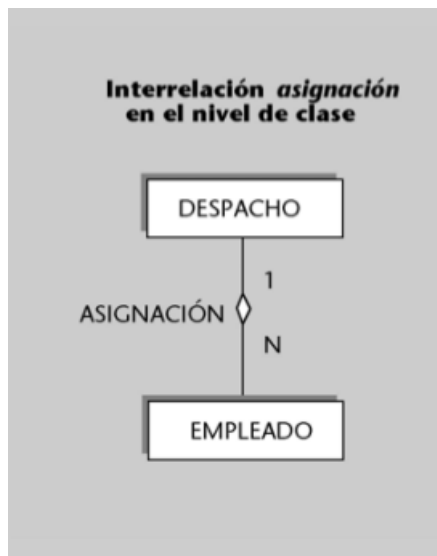


Figura 4.5:

La relación asignación entre la entidad empleado y la entidad despacho tiene conectividad 1:N, y la N está en el lado de la entidad empleado. Esto significa que un empleado tiene un solo despacho asignado, pero que, en cambio, un despacho puede tener uno o más empleados asignados.

3. Conectividad M:N



Figura 4.6:

Para analizar la conectividad M:N, considere la relación evaluación de la figura 4.6 la cual nos indica que un estudiante puede ser evaluado por varias asignaturas y, al mismo tiempo, que una asignatura tiene varios estudiantes por evaluar.

4.4.3 Caso de estudio: base de datos de casas de colonias

En este punto, y antes de continuar explicando construcciones más complejas del modelo ER, puede resultar muy ilustrativo ver la aplicación práctica de las construcciones que hemos estudiado hasta ahora. Por este motivo, analizaremos un caso práctico de diseño con el modelo ER que corresponde a una base de datos destinada a la gestión de las inscripciones en un conjunto de casas de colonias. El modelo ER de esta base de datos será bastante sencillo e incluirá sólo entidades, atributos e relaciones binarias (no incluirá relaciones n-arias ni otros tipos de estructuras).

La descripción siguiente explica con detalle los requisitos de los usuarios que hay que tener en cuenta al hacer el diseño conceptual de la futura base de datos:

1. Cada casa de colonias tiene un nombre que la identifica. Se desea saber de cada una, aparte del nombre, la capacidad (el número de niños que se pueden alojar en cada una como máximo), la comarca donde está situada y las ofertas de actividades que proporciona. Una casa puede ofrecer actividades como natación, esquí, remo, pintura, fotografía, música, etc.
2. Es necesario tener en cuenta que en una casa de colonias se pueden practicar varias actividades (cada casa debe ofrecer como mínimo una), y también puede ocurrir que una misma actividad se pueda llevar a cabo en varias casas. Sin embargo, toda actividad que se registre en la base de datos debe ser ofertada como mínimo en una de las casas.

3. Interesa tener una evaluación de las ofertas de actividades que proporcionan las casas. Se asigna una calificación numérica que indica el nivel de calidad que tiene cada una de las actividades ofertadas.
4. Las casas de colonias alojan niños que se han inscrito para pasar en ellas unas pequeñas vacaciones. Se quiere tener constancia de los niños que se alojan en cada una de las casas para el momento. Se supone que hay casas que están vacías (no se alojan niños) durante algunas temporadas.
5. De los niños que se alojan en alguna de las casas, interesa conocer el código de identificación, su nombre, su apellido, el número de teléfono de sus padres y su comarca de residencia.
6. De las comarcas donde hay casas o donde residen niños, se quiere tener registrados la superficie y el número de habitantes. Considere que puede haber comarcas donde no residen los niños que se alojan en un momento determinado en las casas de colonias, y comarcas que no disponen de casas para alojamiento. La figura 4.7 muestra un diagrama ER que satisface los requisitos anteriores. Los atributos de las entidades no aparecen en el diagrama y se listan aparte.

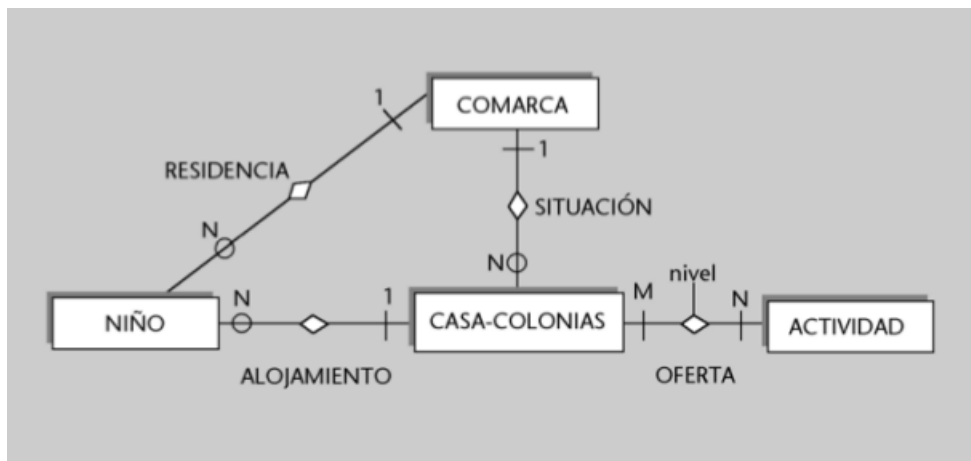


Figura 4.7:

Los atributos de las entidades que figuran en el diagrama son los siguientes (las claves primarias están subrayadas):

```
graph TD; CC[CASA-COLONIAS] --- CC_K[nombre-casa]; CC --- CC_A[capacidad]; A[ACTIVIDAD] --- A_K[nombre-actividad]; N[NIÑO] --- N_K[código-niño]; N --- N_A[nombre]; N --- N_P[apellido]; N --- N_T[teléfono]; C[COMARCA] --- C_K[nombre-comarca]; C --- C_P[superficie]; C --- C_NH[número-habitantes];
```

CASA-COLONIAS
nombre-casa, capacidad

ACTIVIDAD
nombre-actividad

NIÑO
código-niño, nombre, apellido, teléfono

COMARCA
nombre-comarca, superficie, número-habitantes

Figura 4.8:

4.4.4 Aspectos del modelo ER

A continuación, un comentario de los aspectos relevantes de este modelo ER:

1. Una de las dificultades durante la modelización conceptual es decidir si una información determinada es una entidad o un atributo. En el ejemplo, puede resultar difícil decidir si comarca se debe modelizar como una entidad o como un atributo. A primera vista, podría parecer que comarca debe ser un atributo de la entidad casa-colonias para indicar dónde está situada una casa de colonias, y también un atributo de la entidad niño para indicar la residencia del niño. Sin embargo, esta solución no sería adecuada, porque se quieren tener información adicional asociada a la comarca: la superficie y el número de habitantes. Es preciso que comarca sea una entidad para reflejar esta información adicional como atributos de la entidad. La entidad comarca tendrá que estar, evidentemente, relacionada con las entidades niño y casa-colonias. Observe que de este modo, además, se hace patente que las comarcas de residencia de los niños y las comarcas de situación de las casas es información de un mismo tipo.
2. Otra decisión que hay que tomar es si el concepto actividad se modeliza como una entidad o como un atributo. Actividad no tiene información adicional asociada; no tiene, por lo tanto, más atributos que los que forman la clave. Aún así, es necesario que actividad sea una entidad para que, mediante la relación oferta, se indique que una casa de colonias ofrece actividades. Note que las actividades ofertadas no se pueden expresar como un atributo de casa-colonias, porque una casa puede ofrecer muchas actividades y, en este caso, el atributo no podría tomar un valor único.
3. Otra elección difícil, al diseñar un modelo ER, consiste en modelizar una información determinada como una entidad o como una relación. Por ejemplo, podría haber establecido que oferta, en lugar de ser una relación, fuese una entidad; lo habría hecho así:

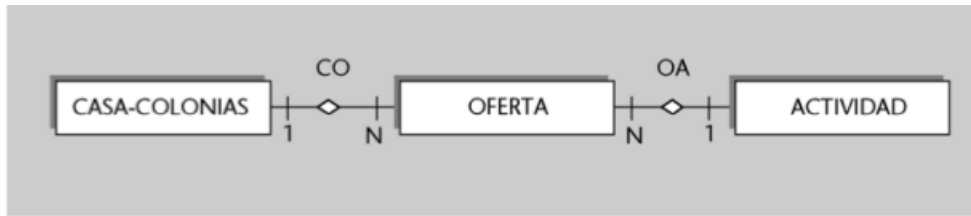


Figura 4.9:

La entidad oferta representada en la figura anterior tiene los atributos que se presentan a continuación:

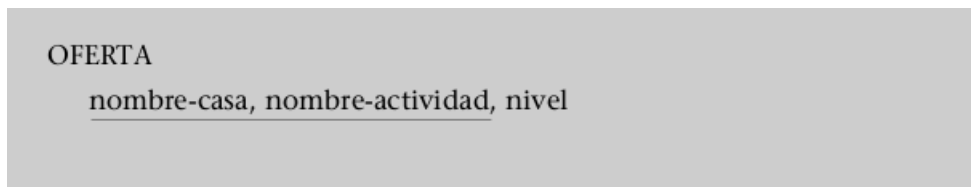


Figura 4.10:

Esta solución no acaba de reflejar adecuadamente la realidad. Si analiza la clave de oferta, puede ver que se identifica con nombre-casa, que es la clave de la entidad casa-colonias, y con nombre-actividad, que es la clave de la entidad actividad. Esto hace sospechar que oferta, de hecho, corresponde a una asociación o relación entre casas y actividades. En consecuencia, refleja la realidad con más exactitud si modeliza oferta como una relación entre estas entidades.

4. Finalmente, durante el diseño conceptual hay que evitar las redundancias. Por ejemplo, si hubiese interrelacionado comarca con actividad para saber qué actividades se realizan en las casas de cada una de las comarcas, habría tenido información redundante. La relación oferta junto con la relación situación hacen saber, de forma indirecta, qué actividades se hacen en las comarcas.

4.5 Ejercicios

1. Escriba un artículo cuyo título tenga 12 palabras, cuerpo tenga 150 palabras y un resumen de 30 palabras para explicar los modelos de base de datos relacional, orientada a objetos, jerárquica, de red.
2. Explique alguna variante del modelo ER.
3. Investigar qué es una entidad débil.
4. Se le solicita al Desarrollador del Banco "mis ahorros" diseñar una base de datos para la gestión del personal que ahí labora. Esta entidad bancaria dispone de muchos empleados y de numerosas sucursales o agencias. La siguiente descripción resume los requerimientos planteados:

- a) Los empleados se identifican por un código de empleado, y también desea conocer su CI, su NSS, su nombre y su apellido. Será importante registrar su ciudad de residencia, considerando que hay ciudades donde no residen empleados.
- b) Interesa saber en qué ciudades están ubicadas las diversas agencias de la entidad bancaria. Estas agencias bancarias se identifican por la ciudad donde están y por un nombre que distingue las agencias de una misma ciudad. Se quiere tener constancia del número de habitantes de las ciudades, así como de la dirección y el número de teléfono de las agencias. Se debe considerar que la base de datos también incluye ciudades donde no hay agencias.
- c) Un empleado, en un momento determinado, trabaja en una sola agencia, lo cual no impide que pueda ser trasladado a otra o, incluso, que vuelva a trabajar en una agencia donde ya había trabajado anteriormente. Se quiere tener constancia del historial del paso de los empleados por las agencias.
- d) Los empleados pueden tener títulos académicos (aunque no todos los tienen). Se quiere saber qué títulos tienen los empleados.
- e) Cada empleado tiene un cargo determinado (auxiliar, oficial de segunda, oficial de primera, etc.). A cada cargo le corresponde un sueldo base determinado y un precio por hora extra también determinado. Se quiere tener constancia del cargo actual de cada empleado, del sueldo base y el precio de la hora extra de cada cargo.
- f) Algunos empleados (no todos) están afiliados a alguna central sindical. Se ha pactado descontar de la nómina mensual la cuota sindical a los afiliados a cada central. Esta cuota es única para todos los afiliados a una central determinada. Es necesario almacenar las afiliaciones a una central de los empleados y las cuotas correspondientes a las diferentes centrales sindicales.
- g) Hay dos tipos de empleados diferentes:
 - 1) Los que tienen contrato fijo, cuya antigüedad quiere conocer. Los que tienen contrato temporal, de los cuales interesa saber las fechas de inicio y finalización de su último contrato. Si un empleado temporal pasa a ser fijo, se le asigna un nuevo código de empleado; considere que un empleado fijo nunca pasa a ser temporal. Todo lo que se ha indicado hasta ahora (traslados, cargo, afiliación sindical, etc.) es aplicable tanto a empleados fijos como a temporales.
- h) Los empleados fijos tienen la posibilidad de pedir diferentes tipos preestablecidos de préstamos (matrimonio, adquisición de vivienda, estudios, etc.), que pueden ser concedidos o no. En principio, no hay límite a la hora de pedir varios préstamos a la vez, siempre que no se pida más de uno del mismo tipo al mismo tiempo. Se quiere registrar los préstamos pedidos por los empleados, y hacer constar si han sido concedidos o no. Cada tipo de préstamo tiene establecidas diferentes condiciones; de éstas, en particular, nos interesa saber el tipo de interés y el periodo de vigencia del préstamo.
- i) Además, se le solicita escribir cada una de las entidades pertenecientes a "mis ahorros" los atributos correspondientes a cada entidad (las claves primarias van subrayadas).

A continuación, algunos comentarios de los aspectos que pueden resultar complejos de este modelo ER:

1. La entidad agencia se ha considerado débil porque su atributo nombre-agencia sólo distingue las agencias situadas en una misma ciudad. Para identificar de forma total una agencia, es

necesario saber en qué ciudad está situada. De este modo, la relación situación es la que hace completa la identificación de la entidad agencia.

2. La relación petición es ternaria y asocia a empleados fijos que hacen peticiones de préstamos, los tipos pedidos por ellos y las fechas en que se hacen estas peticiones.
3. El lado de la entidad fecha se conecta con "muchos" porque un mismo empleado puede pedir un mismo tipo de préstamo varias veces en fechas distintas. La entidad fijo se conecta con "muchos" porque un tipo de préstamo determinado puede ser pedido en una misma fecha por varios empleados. También la entidad tipo-préstamo se conecta con "muchos" porque es posible que un empleado en una fecha determinada pida más de un préstamo de tipo diferente.
4. El atributo concedido/no indica si el estado del préstamo. Es un atributo porque su valor depende al mismo tiempo del empleado fijo que hace la petición, del tipo de préstamo pedido y de la fecha de petición.
5. La relación traslado también es ternaria porque registra el paso de los empleados por las distintas agencias. Un traslado concreto asocia a un empleado, una agencia donde él trabajará y una fecha inicial en la que empieza a trabajar en la agencia. El atributo de la relación fecha-fin indica en qué fecha finaliza su asignación a la agencia (fecha-fin tendrá el valor nulo cuando un empleado trabaja en una agencia en el momento actual y no se sabe cuándo se le trasladará). Observe que fecha-fin debe ser un atributo de la relación. Si se colocase en una de las tres entidades interrelacionadas, no podría ser un atributo univaluado. Conviene observar que esta relación no registra todas y cada una de las fechas en las que un empleado está asignado a una agencia, sino sólo la fecha inicial y la fecha final de la asignación. Es habitual que, para información que son ciertas durante todo un periodo de tiempo, se registre en la base de datos únicamente el inicio y el final del periodo.
Note que la entidad agencia se ha conectado con "uno.^{en} la relación traslado, porque no puede ocurrir que, en una fecha, un empleado determinado sea trasladado a más de una agencia.
6. Finalmente, comente la generalización/especialización de la entidad empleado. Los empleados pueden ser de dos tipos; se quieren registrar propiedades diferentes para cada uno de los tipos y también se requieren algunas propiedades comunes a todos los empleados. Por este motivo, es adecuado utilizar una generalización/especialización.

Capítulo 5

Base de datos, SQL y MariaDB

5.1 Entrando y saliendo de MariaDB

Hemos visto cómo instalar MySQL, y como acceder a su consola. Ahora, como primer ejemplo, crearemos una base de datos sencilla, que llamaremos **ejemplo1**. Esta base de datos contendrá una única tabla, llamada **agenda**, con algunos datos de nuestros amigos. Como es nuestra primera base de datos, no pretendemos que sea perfecta, sino sencilla, así que guardaremos tres datos de cada amigo: el nombre, la dirección y la edad.

Para conectarse al servidor MariaDB desde un terminal linux.

Sintaxis:

```
mysql -u<usuario> -p<contraseña> <nombrasededatos>
```

ejemplo:

```
mysql -umiusuario -pmicontrasena mibasededatos
```

Consideraciones:

- Entre las opciones u y el usuario no debe haber espacios en blanco.
- Entre las opciones p y la contraseña no debe haber espacios en blanco.
- Dejar un espacio entre la contraseña y el nombredelabasededatos.

Para salir de la la consola y linea de comando de MySQL y cerrar la conexión con el servidor debes ejecutar el siguiente comando:

```
mysql> quit
```

5.2 Creación de una base de datos

Para crear la base de datos, la estructura que contiene todo, usaremos **create database**, seguido del nombre que tendrá la base de datos, como ya has visto:

```
CREATE DATABASE ejemplo1;
```

Si, por error, vuelves a teclear la orden, no será un problema. Simplemente obtendrás un mensaje de error que te avisará de no se ha podido crear la base de datos porque ya existe.

Podemos tener varias bases de datos en nuestro SGBD¹, así que cada vez que accedamos a MySQL

¹Sistema Gestor de Bases de Datos

deberemos comenzar por indicar cual de ellas queremos usar, con la orden **use**:

```
USE ejemplo1;
```

5.3 Consultas básicas con una tabla

5.3.1 Creación de una tabla

Una base de datos, en general, estará formada por varios bloques de información llamados **tablas**. En nuestro caso, nuestra única tabla almacenará los datos de nuestros amigos. Por tanto, el siguiente paso será decidir qué datos concretos (lo llamaremos **campos**) guardaremos de cada amigo. Debemos pensar también qué tamaño necesitaremos para cada uno de esos datos, porque al gestor de bases de datos habrá que dárselo bastante cuadrículado. Por ejemplo, podríamos decidir lo siguiente:

```
1 | nombre - texto, hasta 20 letras
2 | dirección - texto, hasta 40 letras
3 | edad - números, de hasta 3 dígitos
```

Cada gestor de bases de datos tendrá una forma de llamar a esos tipos de datos. En MySQL podemos usar **VARCHAR** para referirnos a texto hasta una cierta longitud variable, y **NUMERIC** para números de una determinada cantidad de cifras. Además, en la mayoría de gestores de base de datos, será recomendable (a veces incluso obligatorio) que los nombres de los campos no contengan acentos ni caracteres internacionales, y los nombres de tablas y campos se suelen indicar en minúsculas, de modo que la orden necesaria para crear esta tabla sería:

```
1 | CREATE TABLE personas (
2 | nombre varchar(20),
3 | direccion varchar(40),
4 | edad numeric(3)
5 | );
```

5.3.2 Introducción de datos

Para introducir datos usaremos la orden **INSERT INTO**, e indicaremos tras la palabra **VALUES** los valores de los campos, en el mismo orden en el que se habían definido los campos. Los valores de los campos de texto se deberán indicar entre comillas, y los valores para campos numéricos ni necesitarán comillas, así: `INSERT INTO personas VALUES ('juan', 'su casa', 25);` Si no queremos indicar todos los datos, deberemos detallar qué campos vamos a introducir y en qué orden, así: `INSERT INTO personas (nombre, direccion)VALUES ('Héctor', 'calle 1');` (Como se ve en este ejemplo, los datos sí podrán contener acentos y respetar las mayúsculas y minúsculas donde sea necesario).

De igual modo, también será necesario indicar los campos si se desea introducir los datos en un orden distinto al de definición:

```
INSERT INTO personas (direccion, edad, nombre)VALUES ("Calle 2", 30, "Daniel");
```

(Y como puedes apreciar en este ejemplo, los campos de texto se pueden indicar también entre comillas dobles, si te resulta más cómodo).

5.3.3 Mostrar datos

Para ver los datos almacenados en una tabla usaremos el formato "SELECT campos FROM tabla". Si queremos ver todos los campos, lo indicaremos con un asterisco:

```
SELECT * FROM personas;
```

que, en nuestro caso, daría como resultado:

```
1 | +-----+-----+-----+
2 | | nombre | direccion | edad |
3 | +-----+-----+-----+
4 | | juan   | su casa   | 25   |
5 | | Héctor | calle 1   | NULL |
6 | | Daniel | Calle 2   | 30   |
7 | +-----+-----+-----+
```

Como puedes observar, no habíamos introducido los detalles de la edad de Héctor. No se muestra un cero ni un espacio en blanco, sino que se anota que hay un **valor nulo** (NULL) para ese campo. Usaremos esa peculiaridad más adelante.

Si queremos ver sólo ciertos campos, deberemos detallar sus nombres, en el orden deseado, separados por comas:

```
SELECT nombre, direccion FROM personas;
```

y obtendríamos:

```
1 | +-----+-----+
2 | | nombre | direccion |
3 | +-----+-----+
4 | | juan   | su casa   |
5 | | Héctor | calle 1   |
6 | | Daniel | Calle 2   |
7 | +-----+-----+
```

5.3.4 Buscar por contenido

Normalmente no queremos ver todos los datos que hemos introducido, sino sólo aquellos que cumplan cierta condición. Ésta, se indica añadiendo un apartado **WHERE** a la orden **SELECT**. Por ejemplo, se podrían ver los datos de una persona a partir de su nombre de la siguiente forma:

```
SELECT nombre, direccion FROM personas WHERE nombre = 'juan';
```

```
1 | +-----+-----+
2 | | nombre | direccion |
3 | +-----+-----+
4 | | juan   | su casa   |
5 | +-----+-----+
```

En ocasiones no queremos comparar todo el campo con un texto exacto, sino ver si contiene un cierto texto (por ejemplo, porque sólo sepamos un apellido o parte de la calle). En ese caso, no compararíamos con el símbolo **igual** (=), sino que usaríamos la palabra **LIKE**, y para las partes que no conozcamos usaremos el comodín "%", como en este ejemplo:

```
SELECT nombre, direccion FROM personas WHERE direccion LIKE '\\%calle\\%';
```

que nos diría el nombre y la dirección de nuestros amigos que viven en lugares que contengan la palabra **calle**, aunque ésta pueda estar precedida por cualquier texto (%) y quizá también con otro texto (%) a continuación:

```
1 | +-----+-----+
2 | | nombre | direccion |
3 | +-----+-----+
4 | | Héctor | calle 1   |
5 | | Daniel | Calle 2   |
6 | +-----+-----+
```

5.3.5 Mostrar datos ordenados

Cuando una consulta devuelva muchos resultados, resultará poco legible si éstos se encuentran desordenados. Por eso, será frecuente añadir **ORDER BY** seguido del nombre del campo que se quiere usar para ordenar (por ejemplo **ORDER BY nombre**).

Es posible indicar varios campos de ordenación, de modo que se mire más de un criterio en caso de que un campo coincida (por ejemplo **ORDER BY apellidos, nombre**).

Si se quiere que alguno de los criterios de ordenación se aplique en orden contrario (por ejemplo, textos de la Z a la A o números del valor más grande al más pequeño), se puede añadir la palabra **DESC** (de **descending**, descendiendo) al criterio correspondiente, como en **ORDER BY precio DESC**. Si no se indica esa palabra, se sobreentiende que se quiere obtener los resultados en orden ascendente (lo que indicaría la palabra **ASC**, que es opcional).

```
1 | SELECT nombre, direccion
2 | FROM personas
3 | WHERE direccion LIKE '%calle%'
4 | ORDER BY nombre, direccion DESC;
5 |
6 | +-----+-----+
7 | | nombre | direccion |
8 | +-----+-----+
9 | | Daniel | Calle 2   |
10 | | Héctor | calle 1   |
11 | +-----+-----+
```

5.3.6 Salir de MySQL

Es suficiente por hoy. Para terminar nuestra sesión de MySQL, tecleamos la orden quit o exit y volveremos al sistema operativo.

5.3.7 Ejercicios propuestos

Si no sólo quieres leer, sino que tu intención es aprender y asegurarte de que consolidas los conocimientos, deberías practicar un poco. Aquí tienes una primera tanda de ejercicios propuestos. Si te surgen dudas al resolverlos o quieres asegurarte de que tus respuestas son correctas, consulta con tu profesor.

- Crea una base de datos llamada **ejercicio1**.
- Crea en ella una tabla llamada **ciudades**. De cada ciudad se deseará guardar su nombre (de 40 letras o menos) y su población (de 9 cifras o menos).
- Usando el formato abreviado de **INSERT**, añade la ciudad **Turmero**, con 328.648 habitantes.
- Indicando primero la población y luego el nombre, añade una ciudad de 3.141.991 habitantes llamada **Maracay**.
- Muestra todos los datos de todas las ciudades existentes en la base de datos.
- Muestra la población de la ciudad llamada **Cagua**.
- Muestra el nombre y la población de todas las ciudades que contienen una letra **e** en su nombre.
- Muestra el nombre y la población de todas las ciudades, ordenadas de la más poblada a la menos poblada. Si dos o más ciudades tienen misma cantidad de habitantes, se mostrarán ordenadas de la A a la Z.

5.4 Consultas básicas con dos tablas

5.4.1 Formalizando conceptos

Hay una serie de detalles que hemos pasado por alto y que deberíamos formalizar un poco antes de seguir:

- SQL es un lenguaje de consulta a bases de datos, significa **Structured Query Language** (lenguaje de consulta estructurado).
- MySQL es un **gestor de bases de datos**, es decir, una aplicación informática que se usa para almacenar, obtener y modificar datos (realmente, se les exige una serie de cosas más, como controlar la integridad de los datos o el acceso por parte de los usuarios, pero por ahora nos basta con eso).

En SQL, las órdenes que tecleamos deben terminar en punto y coma (;). Si tecleamos una orden como **SELECT * FROM personas**, sin el punto y coma final, y pulsamos Intro, MySQL responderá mostrando **->** para indicar que todavía no hemos terminado la orden y nos queda algo más por introducir. Precisamente por eso, las órdenes se pueden partir para que ocupen varias líneas, como hemos hecho con **CREATE TABLE**.

Las órdenes de SQL se pueden introducir en mayúsculas o minúsculas. Aún así, es frecuente emplear mayúsculas para las órdenes y minúsculas para los campos y tablas, de modo que la sentencia completa resulte más legible. Así, para mostrar todos los nombres almacenados en nuestra tabla **personas**, se podría hacer con:

```
SELECT nombre FROM personas;
```

5.4.2 Por qué varias tablas?

Puede haber varios motivos:

- Por una parte, podemos tener bloques de información claramente distintos. Por ejemplo, en una base de datos que guarde la información de una empresa tendremos datos como los artículos que distribuimos y los clientes que nos los compran, que no deberían guardarse en una misma tabla.
- Por otra parte, habrá ocasiones en que descubramos que los datos, a pesar de que se podrían clasificar dentro de un mismo **bloque de información** (tabla), serían redundantes: existiría gran cantidad de datos repetitivos, y esto puede dar lugar a dos problemas:
 - Espacio desperdiciado.
 - Posibilidad de errores al introducir los datos, lo que daría lugar a inconsistencias.

Veamos unos datos, que podrían ser parte de una tabla ficticia:

```
1 | +-----+-----+-----+
2 | | nombre | direccion | ciudad |
3 | +-----+-----+-----+
4 | | juan   | su casa  | alicante |
5 | | alberto | calle uno | alicante |
6 | | pedro  | su calle | alicantw |
7 | +-----+-----+-----+
```

Si en vez de repetir **alicante** en cada una de esas fichas, utilizásemos un código de ciudad, por ejemplo **a**, gastaríamos menos espacio (en este ejemplo, 7 bytes menos en cada ficha; con la capacidad de un ordenador actual eso no sería un gran problema).

Por otra parte, hemos tecleado mal uno de los datos: en la tercera ficha no hemos indicado **turmero**, sino **turnerow**, de modo que si hacemos consultas sobre personas de Alicante, la última de ellas no aparecería. Al teclear menos, es también más difícil cometer este tipo de errores.

A cambio, necesitaremos una segunda tabla, en la que guardemos los códigos de las ciudades, y el nombre al que corresponden (por ejemplo: si `códigoDeCiudad = "a"`, la ciudad es **turmero**).

Esta tabla se podría crear así:

```
1 | CREATE TABLE ciudades (
2 |     codigo varchar(3),
3 |     nombre  varchar(30)
4 | );
```

5.4.3 Las claves primarias

Generalmente, y especialmente cuando se usan varias tablas enlazadas, será necesario tener algún dato que nos permita distinguir de forma clara los datos que tenemos almacenados. Por ejemplo, el nombre de una persona no es único: pueden aparecer en nuestra base de datos varios usuarios llamados **Juan López**. Si son nuestros clientes, debemos saber cual es cual, para no cobrar a uno de ellos un dinero que corresponde a otro. Eso se suele solucionar guardando algún dato adicional que sí sea único para cada cliente, como puede ser el Documento Nacional de Identidad, o el Pasaporte. Si no hay ningún dato claro que nos sirva, en ocasiones añadiremos un

código de cliente, inventado por nosotros, o algo similar.

Estos datos que distinguen claramente unas **fichas** de otras los llamaremos **claves primarias**.

Se puede crear una tabla indicando su clave primaria si se añade **PRIMARY KEY** al final de la orden CREATE TABLE, así:

```
1 CREATE TABLE ciudades (  
2     codigo varchar(3),  
3     nombre varchar(30),  
4     PRIMARY KEY (codigo)  
5 );
```

o bien al final de la definición del campo correspondiente, así:

```
1 CREATE TABLE ciudades (  
2     codigo varchar(3) PRIMARY KEY,  
3     nombre varchar(30)  
4 );
```

5.4.4 Creando datos

Comenzaremos creando una nueva base de datos, de forma similar al ejemplo anterior:

```
1 CREATE DATABASE ejemplo2;  
2 USE ejemplo2;
```

Después creamos la tabla de ciudades, que guardará su nombre y su código. Éste será el que actúe como **clave primaria**, para distinguir otra ciudad. Por ejemplo, hay una ciudad llamado **Mariara** en Venezuela, pero también otra en Argentina, otra en Uruguay, dos en Colombia, una en Ohio (Estados Unidos)... el nombre claramente no es único, así que podríamos usar código como **te** para Toledo de España, **ta** para Toledo de Argentina y así sucesivamente.

La forma de crear la tabla con esos dos campos y con esa clave primaria sería:

```
1 CREATE TABLE ciudades (  
2     codigo varchar(3),  
3     nombre varchar(30),  
4     PRIMARY KEY (codigo)  
5 );
```

Mientras que la tabla de personas sería casi igual al ejemplo anterior, pero añadiendo un nuevo dato: el código de la ciudad:

```
1 CREATE TABLE personas (  
2     nombre varchar(20),  
3     direccion varchar(40),  
4     edad decimal(3),  
5     codciudad varchar(3)  
6 );
```

Para introducir datos, el hecho de que exista una clave primaria no supone ningún cambio, salvo por el hecho de que no podemos introducir dos ciudades con el mismo código:

```

1 INSERT INTO ciudades VALUES ('a', 'turmero');
2 INSERT INTO ciudades VALUES ('b', 'cagua');
3 INSERT INTO ciudades VALUES ('m', 'maracay');
4
5 INSERT INTO personas VALUES ('juan', 'su casa', 25, 'a');
6 INSERT INTO personas VALUES ('pedro', 'su calle', 23, 'm');
7 INSERT INTO personas VALUES ('alberto', 'calle uno', 22, 'b');

```

5.4.5 Mostrando datos

Cuando queremos mostrar datos de varias tablas a la vez, deberemos hacer unos pequeños cambios en las órdenes **select** que hemos visto.

En primer lugar, indicaremos varios nombres después de **FROM** (de cada tabla que necesitemos).

Además, puede ocurrir que cada tengamos campos con el mismo nombre en distintas tablas (el nombre de una persona y el nombre de una ciudad), y en ese caso deberemos escribir el nombre de la tabla antes del nombre del campo.

Por eso, una consulta básica sería algo parecido (sólo parecido) a:

```
SELECT personas.nombre, direccion, ciudades.nombre FROM personas, ciudades;
```

Pero esto todavía tiene problemas: estamos combinando **TODOS** los datos de la tabla de personas con **TODOS** los datos de la tabla de ciudades, de modo que obtenemos $3 \times 3 = 9$ resultados:

```

1 +-----+-----+-----+
2 | nombre | direccion | nombre |
3 +-----+-----+-----+
4 | juan   | su casa   | alicante |
5 | pedro  | su calle  | alicante |
6 | alberto | calle uno | alicante |
7 | juan   | su casa   | barcelona |
8 | pedro  | su calle  | barcelona |
9 | alberto | calle uno | barcelona |
10 | juan   | su casa   | madrid |
11 | pedro  | su calle  | madrid |
12 | alberto | calle uno | madrid |
13 +-----+-----+-----+
14 9 rows in set (0.00 sec)

```

Pero esos datos no son reales: si **juan** vive en la ciudad de código **a**, sólo debería mostrarse junto al nombre **alicante**. Nos falta indicar esa condición: el código de ciudad que aparece en la persona debe ser el mismo que el código que aparece en la ciudad', así:

```

1 SELECT personas.nombre, direccion, ciudades.nombre
2 FROM personas, ciudades
3 WHERE personas.codciudad = ciudades.codigo;

```

Esta será la forma en que trabajaremos normalmente. Este último paso se puede evitar en ciertas circunstancias, pero ya lo veremos más adelante. El resultado de esta consulta sería:

```

1 +-----+-----+-----+

```

```

2 | nombre | direccion | nombre |
3 | +-----+-----+-----+
4 | juan   | su casa  | alicante |
5 | alberto | calle uno | barcelona |
6 | pedro  | su calle | madrid   |
7 | +-----+-----+-----+

```

Ese sí es el resultado correcto. Cualquier otra consulta que implique las dos tablas deberá terminar comprobando que los dos códigos coinciden. Por ejemplo, para ver qué personas viven en la ciudad llamada **madrid**, haríamos:

```

1 | SELECT personas.nombre, direccion, edad
2 | FROM personas, ciudades
3 | WHERE ciudades.nombre='madrid'
4 | AND personas.codciudad = ciudades.codigo;
5 |
6 | +-----+-----+-----+
7 | nombre | direccion | edad |
8 | +-----+-----+-----+
9 | pedro  | su calle  | 23   |
10 | +-----+-----+-----+

```

Y para saber las personas de ciudades que comiencen con la letra **b**, haríamos:

```

1 | SELECT personas.nombre, direccion, ciudades.nombre
2 | FROM personas, ciudades
3 | WHERE ciudades.nombre LIKE 'b\%'
4 | AND personas.codciudad = ciudades.codigo;
5 |
6 | +-----+-----+-----+
7 | nombre | direccion | nombre |
8 | +-----+-----+-----+
9 | alberto | calle uno | barcelona |
10 | +-----+-----+-----+

```

Si en nuestra tabla puede haber algún dato que se repita, como la dirección, podemos pedir un listado sin duplicados, usando la palabra **distinct**: `SELECT DISTINCT direccion FROM personas;`

5.4.6 Ejecutando un lote de órdenes

Hasta ahora hemos tecleado todas las órdenes desde dentro del entorno de MySQL, una por una. Tenemos otra opción que también puede ser cómoda: crear un fichero de texto que contenga todas las órdenes (por ejemplo, usando un editor de texto avanzado, como Geany o Notepad++) y cargarlo después desde MySQL. Lo podemos hacer de tres formas:

Usar la orden **source**: desde dentro de MySQL teclearíamos algo como **source ejemplo2.sql**;

Cargar las órdenes justo en el momento de entrar a MySQL, con **mysql -u root <ejemplo2.sql**;

(Pero esta alternativa tiene un problema: se darán los pasos que indiquemos en **ejemplo2.sql** y se abandonará el entorno de MySQL, sin que dé tiempo de comprobar si ha existido algún mensaje de error.)

Preparar las órdenes con el editor avanzado, para luego **copiar y pegar** sobre la consola de MySQL.

5.4.7 Ejercicios propuestos

Ya sabes que deberías practicar antes de seguir avanzando. Aquí tienes una segunda serie de ejercicios propuestos. Si te surgen dudas o quieres exponer tus soluciones, puedes consultar a tu profesor.

1. Crea una base de datos llamada **ejercicio2**. En ella guardaremos información (muy poca) de marcas y modelos de coches. De cada marca almacenaremos el nombre y el país de origen, junto con un código de 3 letras. Para cada modelo anotaremos la marca, el nombre y el segmento al que pertenece (por ejemplo, **urbano**, **compacto**, **familiar**, **todoterreno**, etc.) Usaremos sólo dos tablas: una para marcas y otra para modelos, y sólo usaremos clave principal en **marcas**.
2. Usando el formato detallado de **INSERT**, añade la marca **Ferrari**, con país de origen **Italia**. Su código será **F**. Añade también, con código **SAL**, la marca **Saleen**, de **Estados Unidos**.
3. Con el formato abreviado de **INSERT**, añade el modelo **S7** de **Saleen**, que pertenece al segmento llamado **deportivo**. En el mismo segmento, añade el F40 de Ferrari.
4. Muestra las marcas y modelos de todos los coches del segmento **deportivo**.
5. Muestra marca, país y modelo de todos los vehículos cuya marca comienza por **F**.
6. Muestra país, modelo y segmento de los coches cuyo modelo contenga una letra **S**.

5.5 Borrar información

5.5.1 Qué información hay?

Un primer paso antes de ver cómo borrar información es saber qué información tenemos almacenada.

Podemos saber las bases de datos que hay creadas en nuestro sistema con:

```
1 | SHOW DATABASES;
```

Una vez que estamos trabajando con una base de datos concreta (con la orden **USE**), podemos saber las tablas que contiene con:

```
1 | SHOW TABLES;
```

Y para una tabla concreta, podemos saber los campos (columnas) que la forman con **SHOW COLUMNS FROM**:

```
1 | SHOW COLUMNS FROM personas;
```

Por ejemplo, esto daría como resultado:

1		+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
2		Field		Type		Null		Key		Default		Extra		
3		+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
4		nombre		varchar(20)		YES				NULL				
5		direccion		varchar(40)		YES				NULL				
6		edad		decimal(3,0)		YES				NULL				
7		codciudad		varchar(3)		YES				NULL				
8		+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+

5.5.2 Borrar toda la base de datos

En alguna ocasión, como ahora que estamos practicando, nos puede interesar borrar toda la base de datos. La orden para conseguirlo es **DROP DATABASE**: `DROP DATABASE ejemplo2;` Si esta orden es parte de una secuencia larga de órdenes, que hemos cargado por ejemplo con la orden **source**, puede ocurrir que la base de datos no exista. En ese caso, obtendríamos un mensaje de error y se interrumpiría el proceso en ese punto. Podemos evitarlo añadiendo **IF EXISTS**, para que se borre la base de datos sólo si realmente existe: `DROP DATABASE ejemplo2 IF EXISTS;`

5.5.3 Borrar una tabla

Es más frecuente que creamos alguna tabla de forma incorrecta. La solución razonable es corregir ese error, cambiando la estructura de la tabla, pero todavía no sabemos hacerlo. De momento veremos cómo borrar una tabla. La orden es **DROP TABLE**: `DROP TABLE personas;` Al igual que para las bases de datos, podemos hacer que la tabla se borre sólo cuando realmente existe: `DROP TABLE personas IF EXISTS;`

5.5.4 Borrar datos de una tabla

También podemos borrar los datos que cumplen una cierta condición. La orden es **DELETE FROM**, y con la cláusula **WHERE** indicamos las condiciones que se deben cumplir, de forma similar a como hacíamos en la orden **SELECT**:

```
DELETE FROM personas WHERE nombre = 'juan';
```

Esto borraría todas las personas llamadas **juan** que estén almacenadas en la tabla **personas**.

Cuidado: si no se indica el bloque **WHERE**, no se borrarían los datos que cumplen una condición (porque NO HAY condición), sino TODOS los datos existentes. Si es eso lo que se pretende, una forma más rápida de conseguirlo es usar **TRUNCATE TABLE**:

```
TRUNCATE TABLE personas;
```

5.5.5 Ejercicios propuestos

1. Crea una base de datos llamada **ejercicio4**. En ella guardaremos información de artículos de revistas. De cada revista almacenaremos el nombre, el mes y el año, junto con un código de no más de 8 letras. Para cada artículo anotaremos un código, el título, la revista en la que aparece, la página inicial y la página final (se trata de una relación 1:M, ya que cada revista puede contener varios artículos y cada artículo sólo aparecerá en una revista). Diseña el diagrama Entidad-Relación y crea las tablas.

2. Añade la revista **Byte 9**, del mes 10 de 1984, con código **BY009**. Añade también la revista **PcWorld España 195**, del mes 2 de 2003, con código **PCWE195**.
3. Incluye también los artículos: "The IBM PC AT", con código **.AT**, en la revista **Byte 9**, de la página 108 a la 111. "Database Types", con código **DbTypes**, en la revista **Byte 9**, de la página 138 a la 142. "12 Distribuciones Linux", con código **DistLinux**, en la revista **PCWE195**, de la página 96 a la 109.
4. Muestra todos los artículos, ordenados por año, mes y título.
5. Muestra todos los artículos de revistas **Byte** que contengan la palabra **PC** en su nombre, ordenados por título.
6. Crea una tabla **CopiadeArticulos**, con los mismos campos que la tabla **Articulos**. Usa la orden `INSERT INTO CopiadeArticulos (SELECT * FROM articulos)` para volcar a la nueva tabla todos los datos que existían en la antigua.
7. Borra de **CopiadeArticulos** aquellos artículos que comiencen en páginas por encima de la 120. Muestra los nombres de artículos existentes y su página inicial, ordenados por número de página inicial, para comprobar que el borrado es correcto.
8. Borra de **CopiadeArticulos** aquellos artículos que aparezcan en revistas **Byte**. Muestra los nombres de artículos existentes y el nombre de la revista a la que pertenecen, ordenados por revista y luego por título de artículo, para comprobar que el borrado es correcto.
9. Borra la tabla **CopiadeArticulos** y comprueba que ya no aparece en el sistema.

5.6 Modificar información

5.6.1 Modificación de datos

Ya sabemos borrar datos, pero existe una operación más frecuente que esa (aunque también ligeramente más complicada): modificar los datos existentes. Con lo que sabíamos hasta ahora, podíamos conseguir modificar información: si un dato es incorrecto, podríamos borrarlo y volver a introducirlo, pero esto, obviamente, no es lo más razonable... debería existir alguna orden para cambiar los datos. Efectivamente, la hay. El formato habitual para modificar datos de una tabla es `UPDATE tabla SET campo=nuevoValor WHERE condicion`.

Por ejemplo, si hemos escrito **Alberto** en minúsculas (**alberto**), lo podríamos corregir con:

```
UPDATE personas SET nombre = 'Alberto' WHERE nombre = 'alberto';
```

Y si queremos corregir todas las edades para sumarles un año se haría con

```
UPDATE personas SET edad = edad+1;
```

(al igual que habíamos visto para **select** y para **delete**, si no indicamos la parte del **where**, como en este último ejemplo, los cambios se aplicarán a todos los registros de la tabla).

5.6.2 Ejercicios propuestos sobre modificación de datos

(Los tres primeros ejercicios son idénticos -salvo por el nombre de la base de datos- a los del apartado anterior, así que puedes volver a hacerlos o bien ampliar la estructura que habías creado en el apartado 4).

1. Crea una base de datos llamada **ejercicio5**. En ella guardaremos información de artículos de revistas. De cada revista almacenaremos el nombre, el mes y el año, junto con un código de no más de 8 letras. Para cada artículo anotaremos un código, el título, la revista en la que aparece, la página inicial y la página final (se trata de una relación 1:M, ya que cada revista puede contener varios artículos y cada artículo sólo aparecerá en una revista). Diseña el diagrama Entidad-Relación y crea las tablas.
2. Añade la revista **Byte 9**, del mes 10 de 1984, con código **BY009**. Añade también la revista **PcWorld España 195**, del mes 2 de 2003, con código **PCWE195**.
3. Incluye también los artículos: **The IBM PC AT**, con código **AT**, en la revista **Byte 9**, de la página 108 a la 111. **Database Types**, con código **DbTypes**, en la revista **Byte 9**, de la página 138 a la 142. **12 Distribuciones Linux**, con código **DistLinux**, en la revista **PCWE195**, de la página 96 a la 109.
4. Muestra todos los datos: nombre de revista, mes, año, nombre de artículo, página inicial. Deben aparecer ordenados por nombre de artículo.
5. Corrige el artículo **Database Types (DbTypes)**: no comienza en la página 138 sino en la 137.
6. Cambia el nombre del artículo **12 Distribuciones Linux** para que pase a ser **12 Distribuciones GNU/Linux**.

5.6.3 Modificar la estructura de una tabla

Modificar la estructura de una tabla es algo más complicado: añadir campos, eliminarlos, cambiar su nombre o el tipo de datos. En general, para todo ello se usará la orden **ALTER TABLE**. Vamos a ver las posibilidades más habituales.

Para añadir un campo usaríamos **ADD**:

```
ALTER TABLE ciudades ADD habitantes decimal(7);
```

Si no se indica otra cosa, el nuevo campo se añade al final de la tabla. Si queremos que sea el primer campo, lo indicaríamos añadiendo **first** al final de la orden. También podemos hacer que se añada después de un cierto campo, con **AFTER nombreCampo**.

Podemos modificar el tipo de datos de un campo con **MODIFY**. Por ejemplo, podríamos hacer que el campo **habitantes** no fuera un **decimal** sino un entero largo (**bigint**) con:

```
ALTER TABLE ciudades MODIFY habitantes bigint;
```

Si queremos cambiar el nombre de un campo, debemos usar **CHANGE** (se debe indicar el nombre antiguo, el nombre nuevo y el tipo de datos). Por ejemplo, podríamos cambiar el nombre **habitantes** por **numhabitantes**:

```
ALTER TABLE ciudades CHANGE habitantes numhabitantes bigint;
```

Si queremos borrar algún campo, usaremos **drop column**:

```
ALTER TABLE ciudades DROP COLUMN numhabitantes;
```

Muchas de estas órdenes se pueden encadenar, separadas por comas. Por ejemplo, podríamos borrar dos campos con `alter table ciudades drop column num habitantes, drop column provincia`;

También podríamos cambiar el nombre de una tabla con **RENAME**:

```
ALTER TABLE ciudades RENAME ciudad;
```

Y si hemos olvidado indicar la clave primaria en una tabla, también podemos usar **ALTER TABLE** para detallarla posteriormente:

```
ALTER TABLE ciudades ADD PRIMARY KEY(codigo);
```

5.6.4 Ejercicios propuestos sobre modificación de estructura de tablas

1. En la base de datos llamada **ejercicio5** (o en **ejercicio4**, si estás partiendo de la base de datos anterior), amplía la tabla Revista añadiendo al final de todos los campos un campo adicional de tipo texto: el país en donde se edita (por ejemplo, **México**, **España** o **Estados Unidos**).
2. En la tabla de Artículos, añade el campo Autor (texto), antes del número de página inicial.

5.7 Operaciones matemáticas

5.7.1 Operaciones matemáticas

Desde SQL podemos realizar operaciones a partir de los datos antes de mostrarlos. Podemos mostrar cual era la edad de una persona hace un año, con:

```
SELECT edad-1 FROM personas;
```

Los operadores matemáticos que podemos emplear son los habituales en cualquier lenguaje de programación, ligeramente ampliados: + (suma), - (resta y negación), * (multiplicación), / (división).

La división calcula el resultado con decimales; si queremos trabajar con números enteros, también tenemos los operadores DIV (división entera) y MOD (resto de la división):

```
SELECT 5/2, 5 div 2, 5 mod 2;
```

Daríamos como resultado:

1	+-----+-----+-----+			
2	5/2	5 div 2	5 mod 2	
3	+-----+-----+-----+			
4	2.5000	2	1	
5	+-----+-----+-----+			

Para que resulte más legible, se puede añadir un **alias** a cada **nuevo campo** que se genera al plantear las operaciones matemáticas:

1	SELECT 5/2 divReal, 5 div 2 divEnt, 5 mod 2 resto;			
2	+-----+-----+-----+			
3	divReal	divEnt	resto	
4	+-----+-----+-----+			
5	2.5000	2	1	
6	+-----+-----+-----+			

También podríamos utilizar incluso operaciones a nivel de bits, como las del lenguaje C (>> para desplazar los bits varias posiciones a la derecha, << para la izquierda, | para una suma lógica bit a bit, & para un producto lógico bit a bit y ^ para una operación XOR):

```
SELECT 25 >> 1, 25 << 1, 25 | 10, 25 & 10, 25 ^ 10;
```

que daría

```

1 | +-----+-----+-----+-----+
2 | | 25 >> 1 | 25 << 1 | 25 | 10 | 25 & 10 | 25 ^ 10 |
3 | +-----+-----+-----+-----+
4 | |      12 |      50 |      27 |      8 |      19 |
5 | +-----+-----+-----+-----+
```

5.7.2 Funciones de agregación

También podemos aplicar ciertas funciones matemáticas a todo un conjunto de datos de una tabla. Por ejemplo, podemos saber cual es la edad más baja de entre las personas que tenemos en nuestra base de datos, haríamos:

```
SELECT min(edad)FROM personas;
```

Las funciones de agregación más habituales son:

```

1 | min = mínimo valor
2 | max = máximo valor
3 | sum = suma de los valores
4 | avg = media de los valores
5 | count = cantidad de valores
```

La forma habitual de usar **count** es pidiendo con **count(*)** que se muestren todos los datos que cumplen una condición. Por ejemplo, podríamos saber cuantas personas tienen una dirección que comience por la letra s, así:

```
SELECT count(*)FROM personas WHERE direccion LIKE 's%';
```

5.8 Creando una tabla

Crear la base de datos es la parte más fácil, pero en este momento la base de datos está vacía, como lo indica el comando SHOW TABLES:

```

1 | mysql> SHOW TABLES;
2 | Empty set (0.00 sec)
```

La parte un tanto complicada es decidir la estructura que debe tener nuestra base de datos: qué tablas se necesitan y qué columnas estarán en cada tabla.

En principio, necesitamos una tabla que contenga un registro para cada una de nuestras mascotas. Esta puede ser una tabla llamada mascotas, y debe contener por lo menos el nombre de cada uno de nuestros animalitos. Ya que el nombre en sí no es muy interesante, la tabla debe contener alguna otra información. Por ejemplo, si más de una persona en nuestra familia tiene una mascota, es probable que tengamos que guardar la información acerca de quien es el dueño de cada mascota. Así mismo, también sería interesante contar con alguna información más descriptiva tal como la especie, y el sexo de cada mascota.

¿Y que sucede con la edad?. Esto puede ser también de interés, pero no es una buena idea almacenar este dato en la base de datos. La edad cambia conforme pasa el tiempo, lo cual

significa que debemos de actualizar los registros frecuentemente. En vez de esto, es una mejor idea guardar un valor fijo, tal como la fecha de nacimiento. Entonces, cuando necesitemos la edad, la podemos calcular como la diferencia entre la fecha actual y la fecha de nacimiento. MySQL proporciona funciones para hacer operaciones entre fechas, así que no hay ningún problema.

Al almacenar la fecha de nacimiento en lugar de la edad tenemos algunas otras ventajas:

- Podemos usar la base de datos para tareas tales como generar recordatorios para cada cumpleaños próximo de nuestras mascotas.
- Podemos calcular la edad en relación a otras fechas que la fecha actual.

Por ejemplo, si almacenamos la fecha en que murió nuestra mascota en la base de datos, es fácil calcular que edad tenía nuestro animalito cuando falleció. Es probable que estemos pensando en otro tipo de información que sería igualmente útil en la tabla "mascotas", pero para nosotros será suficiente por ahora contar con información de nombre, propietario, especie, nacimiento y fallecimiento.

Usaremos la sentencia CREATE TABLE para indicar como estarán conformados los registros de nuestras mascotas.

```
1 mysql> CREATE TABLE mascotas(  
2   -> nombre VARCHAR(20), propietario VARCHAR(20),  
3   -> especie VARCHAR(20), sexo CHAR(1), nacimiento DATE,  
4   -> fallecimiento DATE);  
5 Query OK, 0 rows affected (0.02 sec)  
6  
7 mysql>
```

VARCHAR es una buena elección para los campos nombre, propietario, y especie, ya que los valores que almacenarán son de longitud variable. No es necesario que la longitud de estas columnas sea la misma, ni tampoco que sea de 20. Se puede especificar cualquier longitud entre 1 y 255, lo que se considere más adecuado. Si resulta que la elección de la longitud de los campos que hemos hecho no resultó adecuada, MySQL proporciona una sentencia ALTER TABLE que nos puede ayudar a solventar este problema.

El campo sexo puede ser representado en una variedad de formas, por ejemplo, "mz "f", o tal vez "masculinoz "femenino", aunque resulta más simple la primera opción.

El uso del tipo de dato DATE para los campos nacimiento y fallecimiento debe de resultar obvio.

Ahora que hemos creado la tabla, la sentencia SHOW TABLES debe producir algo como:

```
1 mysql> SHOW TABLES;  
2 +-----+  
3 | Tables_in_zoologico |  
4 +-----+  
5 | mascotas |  
6 +-----+  
7 1 row in set (0.00 sec)  
8  
9 mysql>
```

Para verificar que la tabla fué creada como nosotros esperábamos, usaremos la sentencia DESCRIBE:

```

1 |mysql> DESCRIBE mascotas;
2 |-----+-----+-----+-----+
3 | Field | Type | Null | Key | Default | Extra |
4 |-----+-----+-----+-----+
5 | nombre | varchar(20) | YES | | NULL | |
6 | propietario | varchar(20) | YES | | NULL | |
7 | especie | varchar(20) | YES | | NULL | |
8 | sexo | char(1) | YES | | NULL | |
9 | nacimiento | date | YES | | NULL | |
10 | fallecimiento | date | YES | | NULL | |
11 |-----+-----+-----+-----+
12 | 6 rows in set (0.01 sec)
13
14 |mysql>

```

Podemos hacer uso de la sentencia DESCRIBE en cualquier momento, por ejemplo, si olvidamos los nombres ó el tipo de las columnas en la tabla.

5.9 Cargando datos en una tabla

Después de haber creado la tabla, ahora podemos incorporar algunos datos en ella, para lo cual haremos uso de las sentencias INSERT y LOAD DATA.

Supongamos que los registros de nuestras mascotas pueden ser descritos por los datos mostrados en la siguiente tabla.

1	Nombre	Propietario	Especie	Sexo	Nacimiento	Fallecimiento
2	Fluffy	Arnoldo	Gato	f	1999-02-04	
3	Mau Juan	Gato	m	1998-03-17		
4	Buffy	Arnoldo	Perro	f	1999-05-13	
5	FanFan	Benito	Perro	m	2000-08-27	
6	Kaiser	Diana	Perro	m	1998-08-31	1997-07-29
7	Chispa	Omar	Ave	f	1998-09-11	
8	Wicho	Tomás	Ave			
9					2000-02-09	
10	Skim	Benito	Serpiente	m	2001-04-29	

Debemos observar que MySQL espera recibir fechas en el formato YYYY-MM-DD, que puede ser diferente a lo que nosotros estamos acostumbrados.

Ya que estamos iniciando con una tabla vacía, la manera más fácil de poblarla es crear un archivo de texto que contenga un registro por línea para cada uno de nuestros animalitos para que posteriormente carguemos el contenido del archivo en la tabla únicamente con una sentencia.

Por tanto, debemos de crear un archivo de texto "mascotas.txt" que contenga un registro por línea con valores separados por tabuladores, cuidando que el orden de las columnas sea el mismo que utilizamos en la sentencia CREATE TABLE. Para valores que no conozcamos podemos usar valores nulos (NULL). Para representar estos valores en nuestro archivo debemos usar \N.

El archivo mascotas.txt

Para cargar el contenido del archivo en la tabla mascotas, usaremos el siguiente comando:

```
1 |mysql> LOAD DATA LOCAL INFILE "mascotas.txt" INTO TABLE mascotas;
```

La sentencia `LOAD DATA` nos permite especificar cuál es el separador de columnas, y el separador de registros, por default el tabulador es el separador de columnas (campos), y el salto de línea es el separador de registros, que en este caso son suficientes para que la sentencia `LOAD DATA` lea correctamente el archivo "mascotas.txt".

Si lo que deseamos es añadir un registro a la vez, entonces debemos hacer uso de la sentencia `INSERT`. En la manera más simple, debemos proporcionar un valor para cada columna en el orden en el cual fueron listados en la sentencia `CREATE TABLE`. Supongamos que nuestra hermana Diana compra un nuevo hamster nombrado Pelusa. Podemos usar la sentencia `INSERT` para agregar su registro en nuestra base de datos.

```
1 | mysql> REPLACE INTO mascotas -> VALUES('Pelusa','Diana','Hamster','f',  
    | '2000-03-30',NULL);
```

Notar que los valores de cadenas y fechas deben estar encerrados entre comillas. También, con la sentencia `INSERT` podemos insertar el valor `NULL` directamente para representar un valor nulo, un valor que no conocemos. En este caso no se usa `\N` como en el caso de la sentencia `LOAD DATA`.

De este ejemplo, debemos ser capaces de ver que es un poco más la tarea que se tiene que realizar si inicialmente cargamos los registros con varias sentencias `INSERT` en lugar de una única sentencia `LOAD DATA`.

5.10 Recuperando información de una tabla

La sentencia `SELECT` es usada para obtener la información guardada en una tabla. La forma general de esta sentencia es:

```
1 | SELECT LaInformaciónQueDeseamos FROM DeQueTabla WHERE Condición  
    | nASatisfacer
```

Aquí, `LaInformaciónQueDeseamos` es la información que queremos ver. Esta puede ser una lista de columnas, o un `*` para indicar "todas las columnas". `DeQueTabla` indica el nombre de la tabla de la cual vamos a obtener los datos. La cláusula `WHERE` es opcional. Si está presente, la `CondiciónASatisfacer` especifica las condiciones que los registros deben satisfacer para que puedan ser mostrados.

5.10.1 Seleccionando todos los datos

La manera más simple de la sentencia `SELECT` es cuando se recuperan todos los datos de una tabla:

```
1 | mysql> SELECT * FROM mascotas;  
2 | +-----+-----+-----+-----+-----+-----+-----+  
3 | | nombre | propietario | especie | sexo | nacimiento | fallecimiento |  
4 | +-----+-----+-----+-----+-----+-----+-----+  
5 | | Fluffy | Arnoldo | Gato | f | 1999-02-04 | NULL |  
6 | | Mau | Juan | Gato | m | 1998-03-17 | NULL |  
7 | | Buffy | Arnoldo | Perro | f | 1999-05-13 | NULL |
```



```

8 | FanFan | Benito | Perro | m | 2000-08-27 | NULL |
9 | Kaiser | Diana | Perro | m | 1998-08-31 | 1997-07-29 |
10 | Chispa | Omar | Ave | f | 1998-09-11 | NULL |
11 | Wicho | Tomás | Ave | NULL | 2000-02-09 | NULL |
12 | Skim | Benito | Serpiente | m | 2001-04-29 | NULL |
13 | Pelusa | Diana | Hamster | f | 2000-03-30 | NULL |
14 +-----+-----+-----+-----+-----+-----+-----+
15 9 rows in set (0.00 sec)

```

Esta forma del SELECT es útil si deseamos ver los datos completos de la tabla, por ejemplo, para asegurarnos de que están todos los registros después de la carga de un archivo.

Por ejemplo, en este caso que estamos tratando, al consultar los registros de la tabla, nos damos cuenta de que hay un error en el archivo de datos (mascotas.txt): parece que Kaiser ha nacido después de que ha fallecido!. Al revisar un poco el pedigree de Kaiser encontramos que la fecha correcta de nacimiento es el año 1989, no 1998.

Hay por lo menos un par de maneras de solucionar este problema.

Editar el archivo "mascotas.txt" para corregir el error, eliminar los datos de la tabla mascotas con la sentencia DELETE, y cargar los datos nuevamente con el comando LOAD DATA:

```

1 |mysql> DELETE FROM mascotas;
2 |mysql> LOAD DATA LOCAL INFILE "mascotas.txt" INTO TABLE mascotas;

```

Sin embargo, si hacemos esto, debemos ingresar los datos de Pelusa, la mascota de nuestra hermana Diana.

La segunda opción consiste en corregir sólo el registro erróneo con una sentencia UPDATE:

```

1 |mysql> UPDATE mascotas SET nacimiento="1989-08-31" WHERE nombre="
  |Kaiser";

```

Como se mostró anteriormente, es muy fácil recuperar los datos de una tabla completa. Pero típicamente no deseamos hacer esto, particularmente cuando las tablas son demasiado grandes. En vez de ello, estaremos más interesados en responder preguntas particulares, en cuyo caso debemos especificar algunas restricciones para la información que deseamos ver.

5.10.2 Seleccionando registros particulares

Podemos seleccionar sólo registros particulares de una tabla. Por ejemplo, si deseamos verificar el cambio que hicimos a la fecha de nacimiento de Kaiser, seleccionamos sólo el registro de Kaiser de la siguiente manera:

```

1 |mysql> SELECT * FROM mascotas WHERE nombre="Kaiser";
2 |-----+-----+-----+-----+-----+-----+
3 | nombre | propietario | especie | sexo | nacimiento | fallecimiento |
4 |-----+-----+-----+-----+-----+-----+
5 | Kaiser | Diana | Perro | m | 1989-08-31 | 1997-07-29 |
6 |-----+-----+-----+-----+-----+-----+
7 | 1 row in set (0.00 sec)

```

La salida mostrada confirma que el año ha sido corregido de 1998 a 1989.

La comparación de cadenas es normalmente no sensitiva, así que podemos especificar el nombre

como "kaiser", "KAISER", etc. El resultado de la consulta será el mismo.

Podemos además especificar condiciones sobre cualquier columna, no sólo el "nombre". Por ejemplo, si deseamos conocer qué mascotas nacieron después del 2000, tendríamos que usar la columna "nacimiento":

```
1 | mysql> SELECT * FROM mascotas WHERE nacimiento >= "2000-1-1";
2 | +-----+-----+-----+-----+-----+-----+-----+
3 | | nombre | propietario | especie | sexo | nacimiento | fallecimiento
4 | | +-----+-----+-----+-----+-----+-----+-----+
5 | | FanFan | Benito | Perro | m | 2000-08-27 | NULL |
6 | | Wicho | Tomás | Ave | NULL | 2000-02-09 | NULL |
7 | | Skim | Benito | Serpiente | m | 2001-04-29 | NULL |
8 | | Pelusa | Diana | Hamster | f | 2000-03-30 | NULL |
9 | +-----+-----+-----+-----+-----+-----+-----+
10 | 4 rows in set (0.00 sec)
```

Podemos también combinar condiciones, por ejemplo, para localizar a los perros hembras:

```
1 | mysql> SELECT * FROM mascotas WHERE especie="Perro" AND sexo="f";
2 | +-----+-----+-----+-----+-----+-----+-----+
3 | | nombre | propietario | especie | sexo | nacimiento | fallecimiento
4 | | +-----+-----+-----+-----+-----+-----+-----+
5 | | Buffy | Arnoldo | Perro | f | 1999-05-13 | NULL |
6 | +-----+-----+-----+-----+-----+-----+-----+
7 | 1 row in set (0.00 sec)
```

La consulta anterior usa el operador lógico AND. Hay también un operador lógico OR:

```
1 | mysql> SELECT * FROM mascotas WHERE especie = "Ave" OR especie = "
  | Gato";
2 | +-----+-----+-----+-----+-----+-----+-----+
3 | | nombre | propietario | especie | sexo | nacimiento | fallecimiento
4 | | +-----+-----+-----+-----+-----+-----+-----+
5 | | Fluffy | Arnoldo | Gato | f | 1999-02-04 | NULL |
6 | | Mau | Juan | Gato | m | 1998-03-17 | NULL |
7 | | Chispa | Omar | Ave | f | 1998-09-11 | NULL |
8 | | Wicho | Tomás | Ave | NULL | 2000-02-09 | NULL |
9 | +-----+-----+-----+-----+-----+-----+-----+
10 | 4 rows in set (0.00 sec)
```

El operador AND y el operador OR pueden ser intercambiados. Si hacemos esto, es buena idea usar paréntesis para indicar como deben ser agrupadas las condiciones:

```
1 | mysql> SELECT * FROM mascotas WHERE (especie = "Gato" AND sexo = "m")
```

```

2  -> OR (especie = "Perro" AND sexo = "f");
3  +-----+-----+-----+-----+-----+-----+
4  | nombre | propietario | especie | sexo | nacimiento | fallecimiento |
5  |       |           |         |     |            |               |
6  | Mau   | Juan   | Gato   | m   | 1998-03-17 | NULL        |
7  | Buffy | Arnoldo | Perro  | f   | 1999-05-13 | NULL        |
8  +-----+-----+-----+-----+-----+-----+

```

5.10.3 Seleccionando columnas particulares

Si no deseamos ver los registros completos de una tabla, entonces tenemos que usar los nombres de las columnas en las que estamos interesados separándolas por coma. Por ejemplo, si deseamos conocer la fecha de nacimiento de nuestras mascotas, debemos seleccionar la columna "nombre" y "nacimiento":

```

1  mysql> SELECT nombre, nacimiento FROM mascotas;
2  +-----+-----+
3  | nombre | nacimiento |
4  +-----+-----+
5  | Fluffy | 1999-02-04 |
6  | Mau   | 1998-03-17 |
7  | Buffy | 1999-05-13 |
8  | FanFan | 2000-08-27 |
9  | Kaiser | 1989-08-31 |
10 | Chispa | 1998-09-11 |
11 | Wicho  | 2000-02-09 |
12 | Skim   | 2001-04-29 |
13 | Pelusa | 2000-03-30 |
14 +-----+-----+
15 9 rows in set (0.00 sec)

```

Para conocer quién tiene alguna mascota, usaremos la siguiente consulta:

```

1  mysql> SELECT propietario FROM mascotas;
2  +-----+
3  | propietario |
4  +-----+
5  | Arnoldo |
6  | Juan   |
7  | Arnoldo |
8  | Benito |
9  | Diana |
10 | Omar |
11 | Tomás |
12 | Benito |
13 | Diana |
14 +-----+
15 9 rows in set (0.00 sec)

```

Sin embargo, debemos notar que la consulta recupera el nombre del propietario de cada mascota, y algunos de ellos aparecen más de una vez. Para minimizar la salida, agregaremos la palabra clave `DISTINCT`:

```
1 mysql> SELECT DISTINCT propietario FROM mascotas;
2 +-----+
3 | propietario |
4 +-----+
5 | Arnoldo |
6 | Juan |
7 | Benito |
8 | Diana |
9 | Omar |
10 | Tomás |
11 +-----+
12 6 rows in set (0.03 sec)
```

Se puede usar también una cláusula `WHERE` para combinar selección de filas con selección de columnas. Por ejemplo, para obtener la fecha de nacimiento de los perritos y los gatitos, usaremos la siguiente consulta:

```
1 mysql> SELECT nombre, especie, nacimiento FROM mascotas
2   -> WHERE especie = "perro" OR especie = "gato";
3 +-----+-----+-----+
4 | nombre | especie | nacimiento |
5 +-----+-----+-----+
6 | Fluffy | Gato | 1999-02-04 |
7 | Mau | Gato | 1998-03-17 |
8 | Buffy | Perro | 1999-05-13 |
9 | FanFan | Perro | 2000-08-27 |
10 | Kaiser | Perro | 1989-08-31 |
11 +-----+-----+-----+
12 5 rows in set (0.00 sec)
```

5.11 Ordenando registros

Se debe notar en los ejemplos anteriores que las filas regresadas son mostradas sin ningún orden en particular. Sin embargo, frecuentemente es más fácil examinar la salida de una consulta cuando las filas son ordenadas en alguna forma útil. Para ordenar los resultados, tenemos que usar una cláusula `ORDER BY`.

Aquí aparecen algunos datos ordenados por fecha de nacimiento:

```
1 mysql> SELECT nombre, nacimiento FROM mascotas ORDER BY nacimiento;
2 +-----+-----+
3 | nombre | nacimiento |
4 +-----+-----+
5 | Kaiser | 1989-08-31 |
6 | Mau | 1998-03-17 |
7 | Chispa | 1998-09-11 |
8 | Fluffy | 1999-02-04 |
9 | Buffy | 1999-05-13 |
10 | Wicho | 2000-02-09 |
```

```

11 | | Pelusa | 2000-03-30 |
12 | | FanFan | 2000-08-27 |
13 | | Skim | 2001-04-29 |
14 +-----+-----+
15 9 rows in set (0.00 sec)

```

En las columnas de tipo caracter, el ordenamiento es ejecutado normalmente de forma no sensitiva, es decir, no hay diferencia entre mayúsculas y minúsculas. Sin embargo, se puede forzar un ordenamiento sensitivo al usar el operador BINARY.

Para ordenar en orden inverso, debemos agregar la palabra clave DESC al nombre de la columna que estamos usando en el ordenamiento:

```

1  mysql> SELECT nombre, nacimiento FROM mascotas ORDER BY
2  -> nacimiento DESC;
3  +-----+-----+
4  | nombre | nacimiento |
5  +-----+-----+
6  | Skim | 2001-04-29 |
7  | FanFan | 2000-08-27 |
8  | Pelusa | 2000-03-30 |
9  | Wicho | 2000-02-09 |
10 | Buffy | 1999-05-13 |
11 | Fluffy | 1999-02-04 |
12 | Chispa | 1998-09-11 |
13 | Mau | 1998-03-17 |
14 | Kaiser | 1989-08-31 |
15 +-----+-----+
16 9 rows in set (0.00 sec)

```

Podemos ordenar múltiples columnas. Por ejemplo, para ordenar por tipo de animal, y poner al inicio los animalitos más pequeños de edad, usaremos la siguiente consulta:

```

1  mysql> SELECT nombre, especie, nacimiento FROM mascotas
2  -> ORDER BY especie, nacimiento DESC;
3  +-----+-----+-----+
4  | nombre | especie | nacimiento |
5  +-----+-----+-----+
6  | Wicho | Ave | 2000-02-09 |
7  | Chispa | Ave | 1998-09-11 |
8  | Fluffy | Gato | 1999-02-04 |
9  | Mau | Gato | 1998-03-17 |
10 | Pelusa | Hamster | 2000-03-30 |
11 | FanFan | Perro | 2000-08-27 |
12 | Buffy | Perro | 1999-05-13 |
13 | Kaiser | Perro | 1989-08-31 |
14 | Skim | Serpiente | 2001-04-29 |
15 +-----+-----+-----+
16 9 rows in set (0.00 sec)

```

Notar que la palabra clave DESC aplica sólo a la columna nombrada que le precede.

5.12 Cálculos con fechas

MySQL proporciona diversas funciones que se pueden usar para efectuar cálculos sobre fechas, por ejemplo, para calcular edades o extraer partes de una fecha (día, mes, año, etc).

Para determinar la edad de cada una de nuestras mascotas, tenemos que calcular la diferencia de años de la fecha actual y la fecha de nacimiento, y entonces substrair uno si la fecha actual ocurre antes en el calendario que la fecha de nacimiento. Las siguientes consultas muestran la fecha actual, la fecha de nacimiento y la edad para cada mascota.

```
1 mysql> SELECT nombre, nacimiento, CURRENT_DATE,
2   -> (YEAR(CURRENT_DATE) - YEAR(nacimiento))
3   -> - (RIGHT(CURRENT_DATE,5) < RIGHT(nacimiento,5)) AS edad FROM
      mascotas;
4 +-----+-----+-----+-----+
5 | nombre | nacimiento | CURRENT_DATE | edad |
6 +-----+-----+-----+-----+
7 | Fluffy | 1999-02-04 | 2002-12-23 | 3 |
8 | Mau | 1998-03-17 | 2002-12-23 | 4 |
9 | Buffy | 1999-05-13 | 2002-12-23 | 3 |
10 | FanFan | 2000-08-27 | 2002-12-23 | 2 |
11 | Kaiser | 1989-08-31 | 2002-12-23 | 13 |
12 | Chispa | 1998-09-11 | 2002-12-23 | 4 |
13 | Wicho | 2000-02-09 | 2002-12-23 | 2 |
14 | Skim | 2001-04-29 | 2002-12-23 | 1 |
15 | Pelusa | 2000-03-30 | 2002-12-23 | 2 |
16 +-----+-----+-----+-----+
17 9 rows in set (0.01 sec)
```

Aquí, YEAR() obtiene únicamente el año y RIGHT() obtiene los cinco caracteres más a la derecha de cada una de las fechas, que representan el mes y el día (MM-DD). La parte de la expresión que compara los valores MM-DD se evalúa a 1 o 0, y permite ajustar el valor de la edad en el caso de que el valor MM-DD de la fecha actual ocurra antes del valor MM-DD de la fecha de nacimiento.

Dado que la expresión en sí es bastante fea, se ha usado un alias (edad) que es el que aparece como etiqueta en la columna que muestra el resultado de la consulta.

Esta consulta debe trabajar bien, pero el resultado puede ser de alguna manera más útil si las filas son presentadas en algún orden. Para ello haremos uso de la cláusula ORDER BY.

Por ejemplo, para ordenar por nombre, usaremos la siguiente consulta:

```
1 mysql> SELECT nombre, nacimiento, CURRENT_DATE,
2   -> (YEAR(CURRENT_DATE) - YEAR(nacimiento))
3   -> - (RIGHT(CURRENT_DATE,5) < RIGHT(nacimiento,5))
4   -> AS edad FROM mascotas ORDER BY nombre;
5 +-----+-----+-----+-----+
6 | nombre | nacimiento | CURRENT_DATE | edad |
7 +-----+-----+-----+-----+
8 | Buffy | 1999-05-13 | 2002-12-23 | 3 |
9 | Chispa | 1998-09-11 | 2002-12-23 | 4 |
10 | FanFan | 2000-08-27 | 2002-12-23 | 2 |
11 | Fluffy | 1999-02-04 | 2002-12-23 | 3 |
12 | Kaiser | 1989-08-31 | 2002-12-23 | 13 |
13 | Mau | 1998-03-17 | 2002-12-23 | 4 |
```

```

14 | | Pelusa | 2000-03-30 | 2002-12-23 | 2 |
15 | | Skim | 2001-04-29 | 2002-12-23 | 1 |
16 | | Wicho | 2000-02-09 | 2002-12-23 | 2 |
17 +-----+-----+-----+-----+
18 9 rows in set (0.00 sec)

```

Para ordenar por edad en lugar de nombre, únicamente tenemos que usar una cláusula ORDER BY diferente:

```

1  mysql> SELECT nombre, nacimiento, CURRENT_DATE,
2  -> (YEAR(CURRENT_DATE) - YEAR(nacimiento))
3  -> - (RIGHT(CURRENT_DATE,5) < RIGHT(nacimiento,5))
4  -> AS edad FROM mascotas ORDER BY edad;
5  +-----+-----+-----+-----+
6  | nombre | nacimiento | CURRENT_DATE | edad |
7  +-----+-----+-----+-----+
8  | Skim | 2001-04-29 | 2002-12-23 | 1 |
9  | FanFan | 2000-08-27 | 2002-12-23 | 2 |
10 | Wicho | 2000-02-09 | 2002-12-23 | 2 |
11 | Pelusa | 2000-03-30 | 2002-12-23 | 2 |
12 | Fluffy | 1999-02-04 | 2002-12-23 | 3 |
13 | Buffy | 1999-05-13 | 2002-12-23 | 3 |
14 | Mau | 1998-03-17 | 2002-12-23 | 4 |
15 | Chispa | 1998-09-11 | 2002-12-23 | 4 |
16 | Kaiser | 1989-08-31 | 2002-12-23 | 13 |
17 +-----+-----+-----+-----+
18 9 rows in set (0.01 sec)

```

Una consulta similar puede ser usada para determinar la edad que tenía una mascota cuando falleció. Para determinar que animalitos ya fallecieron, la condición es que el valor en el campo fallecimiento no sea nulo (NULL). Entonces, para los registros con valor no-nulo, calculamos la diferencia entre los valores fallecimiento y nacimiento.

```

1  mysql> SELECT nombre, nacimiento, fallecimiento,
2  -> (YEAR(fallecimiento) - YEAR(nacimiento))
3  -> - (RIGHT(fallecimiento,5) < RIGHT(nacimiento,5))
4  -> AS edad FROM mascotas WHERE fallecimiento IS NOT NULL;
5  +-----+-----+-----+-----+
6  | nombre | nacimiento | fallecimiento | edad |
7  +-----+-----+-----+-----+
8  | Kaiser | 1989-08-31 | 1997-07-29 | 7 |
9  +-----+-----+-----+-----+
10 1 row in set (0.01 sec)

```

La consulta usa fallecimiento IS NOT NULL, en vez de fallecimiento <>NULL porque NULL es una valor especial. Esto será explicando más a detalle posteriormente.

¿Qué sucede si deseamos conocer cuáles de nuestras mascotas cumplen años el próximo mes? Para este tipo de cálculos, el año y el día son irrelevantes; simplemente tenemos que extraer el valor del mes en la columna nacimiento. Como se mencionó anteriormente, MySQL proporciona diversas funciones para trabajar y manipular fechas, en este caso haremos uso de la función MONTH(). Para ver como trabaja, vamos a ejecutar una consulta muy simple que muestra tanto el valor de una fecha como el valor que regresa la función MONTH().

```

1  mysql> SELECT nombre, nacimiento, MONTH(nacimiento) FROM mascotas;

```

```

2 | +-----+-----+-----+
3 | | nombre | nacimiento | MONTH(nacimiento) |
4 | +-----+-----+-----+
5 | | Fluffy | 1999-02-04 | 2 |
6 | | Mau | 1998-03-17 | 3 |
7 | | Buffy | 1999-05-13 | 5 |
8 | | FanFan | 2000-08-27 | 8 |
9 | | Kaiser | 1989-08-31 | 8 |
10 | | Chispa | 1998-09-11 | 9 |
11 | | Wicho | 2000-02-09 | 2 |
12 | | Skim | 2001-04-29 | 4 |
13 | | Pelusa | 2000-03-30 | 3 |
14 | +-----+-----+-----+
15 | 9 rows in set (0.00 sec)

```

Encontrar los animalitos cuyo cumpleaños es el próximo mes es muy sencillo. Suponiendo que el mes actual es Abril (valor 4), entonces tenemos que buscar los registros cuyo valor de mes sea 5 (Mayo).

```

1 | mysql> SELECT nombre, nacimiento FROM mascotas WHERE MONTH(nacimiento
2 | ) = 5;
3 | +-----+-----+
4 | | nombre | nacimiento |
5 | | Buffy | 1999-05-13 |
6 | +-----+-----+
7 | 1 row in set (0.00 sec)

```

Aquí habrá por supuesto una complicación si el mes actual es Diciembre. No podemos simplemente agregar uno al número del mes (12) y buscar los registros cuyo mes de nacimiento sea 13 porque dicho mes no existe. En vez de esto, tenemos que buscar los animalitos que nacieron en Enero (mes 1).

Sin embargo, lo mejor es que podemos escribir una consulta que funcione no importando cuál sea el mes actual. La función `DATE_ADD()` nos permite agregar un intervalo de tiempo a una fecha dada. Si agregamos un mes al valor regresado por la función `NOW()`, y entonces extraemos el valor del mes con la función `MONTH()`, el resultado es que siempre obtendremos el mes siguiente.

La consulta que resuelve nuestro problema queda así:

```

1 | mysql> SELECT nombre, nacimiento FROM mascotas -> WHERE MONTH(
2 | nacimiento) = MONTH(DATE_ADD(NOW(), INTERVAL 1 MONTH));

```

5.13 Trabajando con valores nulos

El valor `NULL` puede sorprendernos mientras no hayamos trabajado con él. Conceptualmente, `NULL` significa un valor que hace falta, o un valor desconocido, y es tratado de una manera diferente a otros valores. Para verificar si un valor es `NULL` no podemos usar los operadores de comparación tales como `=`, `>`, o `<`.

Para probar esto ejecutemos la siguiente consulta:

```

1 | mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
2 | +-----+-----+-----+-----+

```



```

3 | 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
4 +-----+-----+-----+-----+
5 | NULL | NULL | NULL | NULL |
6 +-----+-----+-----+-----+
7 1 row in set (0.00 sec)

```

Claramente observamos que no obtenemos resultados con algún significado con estos operadores. Es por ello que tenemos que usar los operadores IS NULL e IS NOT NULL:

```

1 mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
2 +-----+-----+
3 | 1 IS NULL | 1 IS NOT NULL |
4 +-----+-----+
5 | 0 | 1 |
6 +-----+-----+
7 1 row in set (0.00 sec)

```

En MySQL, 0 o NULL significan falso y cualquier otro valor significa verdadero. El valor que se considera verdadero por default es 1.

Cuando se usa un ORDER BY, los valores NULL son siempre ordenados primero, aún cuando se use la cláusula DESC.

5.14 Coincidencia de patrones

MySQL proporciona métodos de coincidencia de patrones basados en SQL estándar, así como también basados en expresiones regulares, de manera similar a las utilerías de Unix tales como vi, grep y sed.

La coincidencia de patrones basada en SQL nos permite usar _ (guión bajo) para un solo caracter y % para un arbitrario número de caracteres. En MySQL, los patrones SQL no son sensibles al uso de mayúsculas y minúsculas.

Es importante notar que no se usan los operadores = o <> cuando se usan los patrones SQL; en su lugar se usan los operadores LIKE y NOT LIKE.

A continuación se presentan algunos ejemplos.

Para encontrar los nombres que comienzan con b:

```

1 mysql> SELECT * FROM mascotas WHERE nombre LIKE "b%";
2 +-----+-----+-----+-----+-----+-----+
3 | nombre | propietario | especie | sexo | nacimiento | fallecimiento |
4 +-----+-----+-----+-----+-----+-----+
5 | Buffy | Arnoldo | Perro | f | 1999-05-13 | NULL |
6 +-----+-----+-----+-----+-----+-----+
7 1 row in set (0.00 sec)

```

Para encontrar los nombres que finalizan con fy:

```

1 mysql> SELECT * FROM mascotas WHERE nombre LIKE "%fy";
2 +-----+-----+-----+-----+-----+-----+

```

```

3 | nombre | propietario | especie | sexo | nacimiento | fallecimiento
4 +-----+-----+-----+-----+-----+-----+
5 | Fluffy | Arnoldo | Gato | f | 1999-02-04 | NULL |
6 | Buffy | Arnoldo | Perro | f | 1999-05-13 | NULL |
7 +-----+-----+-----+-----+-----+-----+
8 2 rows in set (0.00 sec)

```

Para encontrar nombres que contienen una s:

```

1 mysql> SELECT * FROM mascotas WHERE nombre LIKE "%s%";
2 +-----+-----+-----+-----+-----+-----+
3 | nombre | propietario | especie | sexo | nacimiento | fallecimiento
4 +-----+-----+-----+-----+-----+-----+
5 | Kaiser | Diana | Perro | m | 1989-08-31 | 1997-07-29 |
6 | Chispa | Omar | Ave | f | 1998-09-11 | NULL |
7 | Skim | Benito | Serpiente | m | 2001-04-29 | NULL |
8 | Pelusa | Diana | Hamster | f | 2000-03-30 | NULL |
9 +-----+-----+-----+-----+-----+-----+
10 4 rows in set (0.01 sec)

```

El otro tipo de coincidencia de patrones proporcionado por MySQL hace uso de expresiones regulares. Para hacer uso de estos tipos de patrones se tienen que usar los operadores REGEXP y NOT REGEXP (o RLIKE y NOT RLIKE, los cuáles son sinónimos).

Algunas características de las expresiones regulares son.

El caracter punto (.) coincide con cualquier caracter.

Una clase de caracteres [...] coincide con cualquier caracter dentro de los paréntesis cuadrados. Por ejemplo, [abc] coincide con a, b o c. Para nombrar un rango de caracteres, se usa el guión. [a-z] coincide con cualquier letra minúscula, mientras que [0-9] coincide con cualquier dígito.

El caracter asterisco (*) coincide con cero o más instancias de lo que le preceda. Por ejemplo, x* coincide con cualquier número de caracteres x, [0-9]* coincide con cualquier número de dígitos, y .* (punto asterisco) coincide con cualquier cosa.

El patrón coincide si éste ocurre en cualquier parte del valor que está siendo evaluado. (Los patrones SQL coinciden únicamente en los valores completos.)

Para indicar el inicio o el final de un valor que está siendo evaluado se usan los caracteres ^ y \$ respectivamente.

Para demostrar como se usan las expresiones regulares, se van a mostrar los ejemplos presentados anteriormente con el operador LIKE, ahora con el operador REGEXP.

Para encontrar los nombre que inician con b:

```

1 mysql> SELECT * FROM mascotas WHERE nombre REGEXP "^b";
2 +-----+-----+-----+-----+-----+-----+
3 | nombre | propietario | especie | sexo | nacimiento | fallecimiento
4 +-----+-----+-----+-----+-----+-----+

```

```

4 | +-----+-----+-----+-----+-----+-----+
5 | | Buffy | Arnoldo | Perro | f | 1999-05-13 | NULL |
6 | +-----+-----+-----+-----+-----+-----+
7 | 1 row in set (0.01 sec)

```

Antes de la versión 3.23.4 de MySQL, el operador REGEXP era sensible al uso de mayúsculas y minúsculas, así que dependiendo de la versión de MySQL con la que se está trabajando puede que obtengamos o no algún resultado en la consulta anterior. Se puede usar también la siguiente consulta para buscar los nombres que inician con la letra b, no importando si es mayúscula o minúscula.

```

1 | mysql> SELECT * FROM mascotas WHERE nombre REGEXP "[bB]";
2 | +-----+-----+-----+-----+-----+-----+
3 | | nombre | propietario | especie | sexo | nacimiento | fallecimiento
4 | |       |            |         |     |            |
5 | | Buffy | Arnoldo | Perro | f | 1999-05-13 | NULL |
6 | +-----+-----+-----+-----+-----+-----+
7 | 1 row in set (0.00 sec)

```

Desde la versión 3.23.4, para forzar que el operador REGEXP sea sensible al uso de mayúsculas y minúsculas, se tiene que usar la palabra clave BINARY para hacer de una de las cadenas, una cadena binaria. Observar los resultados de la siguientes consultas.

```

1 | mysql> SELECT * FROM mascotas WHERE nombre REGEXP BINARY "^b";
2 | Empty set (0.00 sec)
3 |
4 | mysql> SELECT * FROM mascotas WHERE nombre REGEXP BINARY "^B";
5 | +-----+-----+-----+-----+-----+-----+
6 | | nombre | propietario | especie | sexo | nacimiento | fallecimiento
7 | |       |            |         |     |            |
8 | | Buffy | Arnoldo | Perro | f | 1999-05-13 | NULL |
9 | +-----+-----+-----+-----+-----+-----+
10 | 1 row in set (0.01 sec)

```

Para encontrar los nombres que finalizan con la palabra fy, haremos uso del caracter \$.

```

1 | mysql> SELECT * FROM mascotas WHERE nombre REGEXP "fy$";
2 | +-----+-----+-----+-----+-----+-----+
3 | | nombre | propietario | especie | sexo | nacimiento | fallecimiento
4 | |       |            |         |     |            |
5 | | Fluffy | Arnoldo | Gato | f | 1999-02-04 | NULL |
6 | | Buffy | Arnoldo | Perro | f | 1999-05-13 | NULL |

```

```

7 | +-----+-----+-----+-----+-----+-----+
8 | 2 rows in set (0.00 sec)

```

Para encontrar los nombres que contienen una letra s, la consulta sería:

```

1 | mysql> SELECT * FROM mascotas WHERE nombre REGEXP "s";
2 | +-----+-----+-----+-----+-----+-----+
3 | | nombre | propietario | especie | sexo | nacimiento | fallecimiento
4 | | +-----+-----+-----+-----+-----+-----+
5 | | Kaiser | Diana | Perro | m | 1989-08-31 | 1997-07-29 |
6 | | Chispa | Omar | Ave | f | 1998-09-11 | NULL |
7 | | Skim | Benito | Serpiente | m | 2001-04-29 | NULL |
8 | | Pelusa | Diana | Hamster | f | 2000-03-30 | NULL |
9 | +-----+-----+-----+-----+-----+-----+
10 | 4 rows in set (0.00 sec)

```

5.15 Conteo de filas

Las bases de datos son usadas frecuentemente para responder una pregunta, "¿Con qué frecuencia ocurre un cierto tipo de dato en una tabla?". Por ejemplo, tal vez queremos conocer cuántas mascotas tenemos, o cuántas mascotas tiene cada uno de los propietarios.

Contar el número total de animalitos que tenemos es lo mismo que hacer la siguiente pregunta "¿Cuántas filas hay en la tabla mascotas?za que hay un registro por mascota. La función COUNT() es la que nos ayuda en esta situación.

```

1 | mysql> SELECT COUNT(*) FROM mascotas;
2 | +-----+
3 | | COUNT(*) |
4 | +-----+
5 | | 9 |
6 | +-----+
7 | 1 row in set (0.00 sec)

```

Si deseamos conocer cuántas mascotas tiene cada uno de los propietarios, la consulta es la siguiente:

```

1 | mysql> SELECT propietario, COUNT(*) FROM mascotas GROUP BY
2 | propietario ;
3 | +-----+-----+
4 | | propietario | COUNT(*) |
5 | +-----+-----+
6 | | Arnoldo | 2 |
7 | | Benito | 2 |
8 | | Diana | 2 |
9 | | Juan | 1 |
10 | | Omar | 1 |
11 | | Tomás | 1 |
12 | +-----+-----+

```

```
12 | 6 rows in set (0.00 sec)
```

Se debe notar que se ha usado una cláusula GROUP BY para agrupar todos los registros de cada propietario. Si no hacemos esto, obtendremos un mensaje de error:

```
1 | mysql> SELECT propietario, COUNT(*) FROM mascotas;
2 | ERROR 1140: Mixing of GROUP columns (MIN(),MAX(),COUNT()...) with no
3 | GROUP columns is illegal if there is no GROUP BY clause
```

En efecto, el uso de la función COUNT() en conjunto con la cláusula GROUP BY es muy útil en diversas situaciones. A continuación se muestran algunos ejemplos.

El número de animalitos por especie:

```
1 | mysql> SELECT especie, COUNT(*) FROM mascotas GROUP BY especie ;
2 | +-----+-----+
3 | | especie | COUNT(*) |
4 | +-----+-----+
5 | | Ave | 2 |
6 | | Gato | 2 |
7 | | Hamster | 1 |
8 | | Perro | 3 |
9 | | Serpiente | 1 |
10 | +-----+-----+
11 | 5 rows in set (0.00 sec)
```

El número de animalitos por sexo:

```
1 | mysql> SELECT sexo, COUNT(*) FROM mascotas GROUP BY sexo:
2 | +-----+-----+
3 | | sexo | COUNT(*) |
4 | +-----+-----+
5 | | NULL | 1 |
6 | | f | 4 |
7 | | m | 4 |
8 | +-----+-----+
9 | 3 rows in set (0.01 sec)
```

El número de animalitos por combinación de especie y sexo:

```
1 | mysql> SELECT especie, sexo, COUNT(*) FROM mascotas GROUP BY especie,
2 | sexo ;
3 | +-----+-----+-----+
4 | | especie | sexo | COUNT(*) |
5 | +-----+-----+-----+
6 | | Ave | NULL | 1 |
7 | | Ave | f | 1 |
8 | | Gato | f | 1 |
9 | | Gato | m | 1 |
10 | | Hamster | f | 1 |
11 | | Perro | f | 1 |
12 | | Perro | m | 2 |
13 | | Serpiente | m | 1 |
14 | +-----+-----+-----+
15 | 8 rows in set (0.00 sec)
```

No es necesario que se obtengan todos los datos de una tabla cuando se usa la función COUNT(). Por ejemplo, en la consulta anterior, para ver únicamente los datos de perritos y gatitos, la consulta queda de la siguiente manera:

```
1 mysql> SELECT especie, sexo, COUNT(*) FROM mascotas -> WHERE especie
   = "Perro" OR especie="Gato" -> GROUP BY especie, sexo;
2 +-----+-----+-----+
3 | especie | sexo | COUNT(*) |
4 +-----+-----+-----+
5 | Gato   | f   | 1        |
6 | Gato   | m   | 1        |
7 | Perro  | f   | 1        |
8 | Perro  | m   | 2        |
9 +-----+-----+-----+
10 4 rows in set (0.00 sec)
```

O bien, si deseamos el número de animalitos por sexo, y cuyo sexo es conocido:

```
1 mysql> SELECT especie, sexo, COUNT(*) FROM mascotas
2   -> WHERE sexo IS NOT NULL
3   -> GROUP BY especie, sexo ;
4 +-----+-----+-----+
5 | especie | sexo | COUNT(*) |
6 +-----+-----+-----+
7 | Ave     | f   | 1        |
8 | Gato    | f   | 1        |
9 | Gato    | m   | 1        |
10 | Hamster | f   | 1        |
11 | Perro   | f   | 1        |
12 | Perro   | m   | 2        |
13 | Serpiente | m | 1        |
14 +-----+-----+-----+
15 7 rows in set (0.00 sec)
```

5.16 Usando más de una tabla

La tabla mascotas nos ha servido hasta este momento para tener guardados los datos acerca de los animalitos que tenemos. Si deseamos guardar algún otro tipo de información acerca de ellos, tal como los eventos en sus vidas -visitas al veterinario, nacimientos de una camada, etc- necesitaremos de otra tabla. ¿Cómo deberá estar conformada esta tabla?. Lo que necesitamos es:

- El nombre de la mascota para saber a cuál de ellas se refiere el evento.
- Una fecha para saber cuando ocurrió el evento.
- Una descripción del evento.
- Un campo que indique el tipo de evento, si deseamos categorizarlos.

Dadas estas condiciones, la sentencia para crear la tabla eventos queda de la siguiente manera:

```
1 mysql> CREATE TABLE eventos(nombre varchar(20), fecha date,
2   -> tipo varchar(15), descripcion varchar(255));
3 Query OK, 0 rows affected (0.03 sec)
```

De manera similar a la tabla mascotas, es más fácil cargar los datos de los registros iniciales al crear un archivo de texto delimitado por tabuladores en el que se tenga la siguiente información:

```

1 | nombre  fecha tipo  descripción
2 | Fluffy  2001-05-15  camada  4 gatitos, 3 hembras, 1 macho
3 | Buffy  2001-06-23  camada  5 perritos, 2 hembras, 3 machos
4 | Buffy  2002-06-19  camada  2 perritos, 1 hembra, 1 macho
5 | Chispa  2000-03-21  veterinario Una pata lastimada
6 | FanFan  2001-08-27  cumpleaños Primera vez que se enfermó de la
   |         panza
7 | FanFan  2002-08-03  veterinario Dolor de panza
8 | Whicho  2001-02-09  cumpleaños Remodelación de casa El archivo
   |         eventos.txt

```

Cargamos los datos en este archivo con la siguiente sentencia:

```

1 | mysql> LOAD DATA LOCAL INFILE "eventos.txt" INTO TABLE eventos;
2 | Query OK, 7 rows affected (0.02 sec)
3 | Records: 7 Deleted: 0 Skipped: 0 Warnings: 0

```

Tomando en cuenta lo que hemos aprendido en la ejecución de consultas sobre la tabla mascotas, debemos de ser capaces de recuperar algunos datos de la tabla eventos; los principios son los mismos. Sin embargo puede suceder que la tabla eventos por sí misma sea insuficiente para darnos las respuestas que necesitamos.

Supongamos que deseamos conocer la edad de cada mascota cuando tuvieron una camada. La tabla eventos indica cuando ocurrió dicho evento, pero para calcular la edad de la madre, necesitamos sin duda su fecha de nacimiento. Dado que este dato está almacenado en la tabla mascotas, necesitamos de ambas tablas para realizar esta consulta.

```

1 | mysql> SELECT mascotas.nombre,
2 |     -> (TO_DAYS(fecha) - TO_DAYS(nacimiento))/365 AS edad,
3 |     -> descripcion FROM mascotas, eventos
4 |     -> WHERE mascotas.nombre=eventos.nombre
5 |     -> AND tipo='camada';
6 | +-----+-----+-----+
7 | | nombre | edad | descripcion |
8 | +-----+-----+-----+
9 | | Fluffy | 2.28 | 4 gatitos, 3 hembras, 1 macho |
10 | | Buffy | 2.12 | 5 perritos, 2 hembras, 3 machos |
11 | | Buffy | 3.10 | 2 perritos, 1 hembra, 1 macho |
12 | +-----+-----+-----+
13 | 3 rows in set (0.05 sec)

```

Hay diversas cosas que notar acerca de esta consulta: La cláusula FROM lista dos tablas dado que la consulta necesita información que se encuentra en ambas tablas.

Cuando se combina (junta) información de múltiples tablas, es necesario especificar los registros de una tabla que pueden coincidir con los registros en la otra tabla. En nuestro caso, ambas columnas tienen una columna "nombre". La consulta usa la cláusula WHERE para obtener los registros cuyo valor en dicha columna es el mismo en ambas tablas.

Dado que la columna "nombre" ocurre en ambas tablas, debemos de especificar a cuál de las columnas nos referimos. Esto se hace al anteponer el nombre de la tabla al nombre de la columna.

Nota: La función TO_DAYS() regresa el número de días transcurridos desde el año 0 hasta la fecha dada.

No es necesario que se tengan dos tablas diferentes para que se puedan juntar. Algunas veces es útil juntar una tabla consigo misma si se desean comparar registros de la misma tabla. Por ejemplo, para encontrar las posibles parejas entre nuestras mascotas de acuerdo a la especie, la consulta sería la siguiente:

```

1 |mysql> SELECT m1.nombre, m1.sexo, m2.nombre, m2.sexo, m1.especie
2 |-> FROM mascotas AS m1, mascotas AS m2
3 |-> WHERE m1.especie=m2.especie AND m1.sexo="f" AND m2.sexo="m";
4 |-----+-----+-----+-----+-----+
5 | nombre | sexo | nombre | sexo | especie |
6 |-----+-----+-----+-----+-----+
7 | Fluffy | f | Mau | m | Gato |
8 | Buffy | f | FanFan | m | Perro |
9 | Buffy | f | Kaiser | m | Perro |
10 |-----+-----+-----+-----+-----+
11 | 3 rows in set (0.00 sec)

```

En esta consulta se ha especificado un alias para el nombre de la tabla, y es éste el que se utiliza para referirse a las columnas.

5.16.1 Ejercicios propuestos

Estos ejercicios parten de la base de datos **ejercicio5** o **ejercicio4**.

1. Muestra el nombre de cada artículo y la cantidad de páginas que ocupa (página final - página inicial + 1; por ejemplo, un artículo que empiece en la página 3 y acabe en la 4 ocupa dos páginas).
2. Muestra la cantidad de artículos que hay en la base de datos.
3. Muestra la cantidad de revistas que contienen **byte** en su nombre.
4. Muestra la media del año de publicación de las revistas de la base de datos.
5. Muestra el año de publicación de la revista más antigua de la base de datos (el mínimo de los años) y el de la más moderna (el máximo).
6. Muestra el total de páginas que tenemos indexadas (mira la pregunta 1 para ver una pista de cómo calcular las páginas de cada artículo).

5.17 Operadores

MySQL dispone de multitud de operadores diferentes para cada uno de los tipos de columna. Esos operadores se utilizan para construir expresiones que se usan en cláusulas ORDER BY y HAVING de la sentencia SELECT y en las cláusulas WHERE de las sentencias SELECT, DELETE y UPDATE. Además se pueden emplear en sentencias SET.

5.17.1 Operador de asignación

En MySQL creamos variables y las usamos en expresiones.

Para crear una variable hay dos posibilidades. La primera consiste en usar la sentencia SET de este modo:


```

1 | mysql> SET @hoy = CURRENT_DATE();
2 | Query OK, 0 rows affected (0.02 sec)
3 |
4 | mysql> SELECT @hoy;
5 | +-----+
6 | | @hoy      |
7 | +-----+
8 | | 2005-03-23 |
9 | +-----+
10 | 1 row in set (0.00 sec)

```

La otra alternativa permite definir variables de usuario dentro de una sentencia SELECT:

```

1 | mysql> SELECT @x:=10;
2 | +-----+
3 | | @x:=10 |
4 | +-----+
5 | |      10 |
6 | +-----+
7 | 1 row in set (0.00 sec)
8 |
9 | mysql> SELECT @x;
10 | +-----+
11 | | @x      |
12 | +-----+
13 | |      10 |
14 | +-----+
15 | 1 row in set (0.00 sec)

```

En esta segunda forma es donde se usa el operador de asignación :=. Otros ejemplos del uso de variables de usuario pueden ser:

```

1 | mysql> SELECT @fecha_min:=MIN(fecha), @fecha_max:=MAX(fecha) FROM
2 | gente;
3 | +-----+-----+-----+
4 | | @fecha_min:=MIN(fecha) | @fecha_max:=MAX(fecha) |
5 | +-----+-----+-----+
6 | | 1978-06-15          | 2001-12-02          |
7 | +-----+-----+-----+
8 | 1 row in set (0.00 sec)
9 |
10 | mysql> SELECT * FROM gente WHERE fecha=@fecha_min;
11 | +-----+-----+
12 | | nombre  | fecha      |
13 | +-----+-----+
14 | | Mengano | 1978-06-15 |
15 | | Pimplano | 1978-06-15 |
16 | +-----+-----+
17 | 2 rows in set (0.00 sec)

```

Una variable sin asignar será de tipo cadena y tendrá el valor NULL.

5.17.2 Operadores lógicos

Los operadores lógicos se usan para expresiones lógicas complejas. Facilitan el uso de álgebra booleana, y ayudan a crear condiciones mucho más precisas.

En el álgebra booleana sólo existen dos valores posibles para los operandos y los resultados: verdadero y falso. MySQL dispone de dos constantes para esos valores: TRUE y FALSE, respectivamente.

MySQL añade un tercer valor: desconocido. Esto es para que sea posible trabajar con valores NULL. El valor verdadero se implementa como 1 o TRUE, el falso como 0 o FALSE y el desconocido como NULL.

```
1 |mysql> SELECT TRUE, FALSE, NULL;
2 |-----+-----+-----+
3 | TRUE | FALSE | NULL |
4 |-----+-----+-----+
5 |      1 |      0 | NULL |
6 |-----+-----+-----+
7 | 1 row in set (0.00 sec)
```

5.17.3 Operador Y

En MySQL se usan tanto la forma AND como &&, es decir, ambas formas se refieren al mismo operador: Y lógico.

Se trata de un operador diádico o binario, es decir, requiere de dos operandos. El resultado es verdadero sólo si ambos operandos son verdaderos, y falso si cualquier operando es falso. Esto se representa mediante la siguiente tabla de verdad:

A	B	A AND B
falso	falso	falso
falso	verdadero	falso
verdadero	falso	falso
verdadero	verdadero	verdadero
falso	NULL	falso
NULL	falso	falso
verdadero	NULL	NULL
NULL	verdadero	NULL

Al igual que todos los operadores binarios que veremos, el operador Y se puede asociar, es decir, se pueden crear expresiones como A AND B AND C. El hecho de que se requieran dos operandos significa que las operaciones se realizan tomando los operandos dos a dos, y estas expresiones se evalúan de izquierda a derecha. Primero se evalúa A AND B, y el resultado, R, se usa como primer operando de la siguiente operación R AND C.

```
1 |mysql> SELECT 1 AND 0, 1 AND NULL, 0 AND NULL, 1 AND 0 AND 1;
2 |-----+-----+-----+-----+
3 | 1 AND 0 | 1 AND NULL | 0 AND NULL | 1 AND 0 AND 1 |
4 |-----+-----+-----+-----+
5 |      0 |      NULL |      0 |      0 |
6 |-----+-----+-----+-----+
```

```
7 | 1 row in set (0.00 sec)
```

5.17.4 Operador O

En MySQL este operador también tiene dos formas equivalentes OR y ||.

El operador O también es binario. Si ambos operandos son distintos de NULL y el resultado es verdadero si cualquiera de ellos es verdadero, y falso si ambos son falsos. Si uno de los operandos es NULL el resultado es verdadero si el otro es verdadero, y NULL en el caso contrario. La tabla de verdad es:

A	B	A OR B
falso	falso	falso
falso	verdadero	verdadero
verdadero	falso	verdadero
verdadero	verdadero	verdadero
falso	NULL	NULL
NULL	falso	NULL
verdadero	NULL	verdadero
NULL	verdadero	verdadero

```
1 | mysql> SELECT 1 OR 0, 1 OR NULL, 0 OR NULL, 1 OR 0 OR 1;  
2 | +-----+-----+-----+-----+  
3 | | 1 OR 0 | 1 OR NULL | 0 OR NULL | 1 OR 0 OR 1 |  
4 | +-----+-----+-----+-----+  
5 | |      1 |      1 |     NULL |      1 |  
6 | +-----+-----+-----+-----+  
7 | 1 row in set (0.00 sec)
```

Operador O exclusivo XOR también es un operador binario, que devuelve NULL si cualquiera de los operandos es NULL. Si ninguno de los operandos es NULL devolverá un valor verdadero si uno de ellos es verdadero, y falso si ambos son verdaderos o ambos falsos. La tabla de verdad es:

A	B	A XOR B
falso	falso	falso
falso	verdadero	verdadero
verdadero	falso	verdadero
verdadero	verdadero	falso
falso	NULL	NULL
NULL	falso	NULL
verdadero	NULL	NULL
NULL	verdadero	NULL

```
1 | mysql> SELECT 1 XOR 0, 1 XOR NULL, 0 XOR NULL, 1 XOR 0 XOR 1;  
2 | +-----+-----+-----+-----+  
3 | | 1 XOR 0 | 1 XOR NULL | 0 XOR NULL | 1 XOR 0 XOR 1 |  
4 | +-----+-----+-----+-----+  
5 | |      1 |     NULL |     NULL |      0 |
```

```

6 | +-----+-----+-----+-----+
7 | 1 row in set (0.00 sec)

```

5.17.5 Operador de negación

El operador NOT, que también se puede escribir como !, es un operador unitario, es decir sólo afecta a un operando. Si el operando es verdadero devuelve falso, y viceversa. Si el operando es NULL el valor devuelto también es NULL.

A	NOT A
falso	verdadero
verdadero	falso
NULL	NULL

```

1 | mysql> SELECT NOT 0, NOT 1, NOT NULL;
2 | +-----+-----+-----+
3 | | NOT 0 | NOT 1 | NOT NULL |
4 | +-----+-----+-----+
5 | |      1 |      0 |      NULL |
6 | +-----+-----+-----+
7 | 1 row in set (0.02 sec)

```

5.18 funciones en bases de datos

Cuando gestionamos bases de datos tenemos diversas funciones para ver o presentar los reportes de manera organizada, sencilla y adecuada a la necesidad.

En MYSQL existen variadas funciones para gestionar los datos almacenados dentro de ella. En esta oportunidad vamos a revisar algunas funciones que nos ayudarán a esta tarea de gestión en MySQL. Estas funciones son:

Left	Con esta función obtenemos los caracteres a la izquierda.
Substring	Con esta función obtenemos la palabra tomando en cuenta el origen que le indiquemos. Si tenemos una palabra con siete (7) palabras y solicitamos que muestre desde el segundo carácter en adelante.
Concat	Esta función concatena una serie de palabras o parámetros.
Upper	Establece un carácter en mayúscula.
Lower	Establece un carácter en minúscula.

5.18.1 función left

La finalidad que vamos a hacer con la función left es separar la primera letra del resto de los caracteres y convertirla en mayúscula. Para realizar este proceso ingresaremos la siguiente sintaxis:

```
SELECT left(columna, cantidad de espacios)FROM tabla;
```

En nuestro caso ingresaremos lo siguiente:

```
SELECT left(title,1), amount FROM solvetic.solvetic_mysql;
```

Con esto indicamos que la función left deje la primera letra (1) para su edición, este es el resultado

5.18.2 función Upper

Teniendo en mente el caso anterior, la idea es convertir esa primera letra en mayúscula, razón por la cual usaremos la función Upper. La sintaxis que debemos usar es la siguiente:

```
SELECT upper(left(columna, cantidad de espacios))FROM tabla;
```

En nuestro ejemplo debe ser de la siguiente manera:

```
SELECT upper(left(title,1)), amount FROM solvetic.solvetic_mysql;
```

5.18.3 función substring

Ahora nos centraremos en la función substring, recordemos que con ella vemos la cantidad de palabras a partir de un número determinado. La sintaxis a usar es la siguiente:

```
SELECT substring(columna, caracter de partida)FROM tabla;
```

En nuestro ejemplo usaremos la siguiente sintaxis:

```
SELECT substring(title,2), amount FROM solvetic.solvetic_mysql;
```

Esto es importante ya que podemos tener palabras completamente en mayúscula y otras en minúscula.

5.18.4 función Lower

Siguiendo con el ejemplo anterior tenemos algunas palabras en mayúscula lo cual no es el propósito y es allí donde la función lower nos ayuda ya que ella convierte una letra mayúscula en minúscula. La sintaxis a usar es la siguiente: (En este ejemplo):

```
SELECT lower(substring(columna, caracter de partida))FROM tabla;
```

Para ver el resultado ingresamos lo siguiente:

```
SELECT lower(substring(title,2)), amount FROM solvetic.solvetic_mysql;
```

El resultado que obtenemos es todas las letras en minúscula.

5.18.5 función Concat

Como bien sabemos la función Concatenar nos permite unir una o más palabras en una sola y en MySQL no es la excepción. Hemos convertido la letra inicial de nuestras palabras en mayúsculas y todas las demás en minúsculas pero ahora debemos unir estas dos para que el efecto sea el deseado, una presentación limpia y ordenada. En este caso la función concat deberá combinar las siguientes funciones:

```
1 | SELECT upper(left(title,1)), amount FROM solvetic.solvetic_mysql;  
2 | SELECT lower(substring(title,2)), amount FROM solvetic.solvetic_mysql  
   | ;
```

La sintaxis para el uso de concat es la siguiente:

```
SELECT concat(parámetro 1,parámetro 2)FROM tabla;
```

Vemosr que los registros de la columna titles están organizados de la manera indicada.

Si notamos en la cabecera de la columna se ve reflejada toda la sintaxis de la función lo cual es un poco feo para el reporte, podemos asignar el nombre deseado usando la siguiente sintaxis:

```
SELECT concat(parámetro 1,parámetro 2)nombre_cabecera FROM tabla;
```

De esta manera la función concat nos ayuda a organizar mejor nuestros registros en MySQL.

5.19 Actualizar la base de datos

Una vez hayamos realizado cualquier modificación usando alguna de estas funciones debemos actualizar la tabla principal, para ello vamos a usar la siguiente sintaxis:

```
update nombre_tabla set
```

```
Columna_modificada= concat(parámetro 1,parámetro 2)
```

Para nuestro ejemplo es lo siguiente:

```
1 | update solvetic.solvetic_mysql set  
2 | title= concat(upper(left(title,1)),lower(substring(title,2)))
```

De esta manera se actualizará la tabla principal.

Hemos visto como usamos estas interesantes funciones en MySQL para que los registros que tenemos almacenados en la BD se vean de una manera organizada y ordenada para llevar un control más específico de la misma.

5.20 Agrupamiento de Registros y funciones agregadas

5.20.1 GROUP BY

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluye una función SQL agregada, como por ejemplo Sum o Count, en la instrucción SELECT. Su sintaxis es:

```
SELECTcampos FROM tabla WHERE criterio GROUP BY campos del grupo
```

GROUP BY es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción SELECT. Los valores Null en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas. Se utiliza la cláusula WHERE para excluir aquellas filas que no desea agrupar, y la cláusula HAVING para filtrar los registros una vez agrupados.

A menos que contenga un dato Memo u Objeto OLE , un campo de la lista de campos GROUP BY puede referirse a cualquier campo de las tablas que aparecen en la cláusula FROM, incluso si el campo no esta incluido en la instrucción SELECT, siempre y cuando la instrucción SELECT incluya al menos una función SQL agregada.

Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada.

```
SELECT Id_Familia, Sum(Stock)FROM Productos GROUP BY Id_Familia;
```

Una vez que GROUP BY ha combinado los registros, HAVING muestra cualquier registro agrupado por la cláusula GROUP BY que satisfaga las condiciones de la cláusula HAVING.

HAVING es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuales de ellos se van a mostrar.

```
1 | SELECT Id_Familia Sum(Stock) FROM Productos GROUP BY Id_Familia  
2 | HAVINGSum(Stock) > 100 AND NombreProducto Like BOS*;
```

5.20.2 AVG

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente :

Avg(expr)

En donde expr representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por Avg es la media aritmética (la suma de los valores dividido por el número de valores). La función Avg no incluye ningún campo Null en el cálculo.

```
SELECT Avg(Gastos)AS Promedio FROM Pedidos WHERE Gastos > 100;
```

5.20.3 Count

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente:

Count(expr)

En donde expr contiene el nombre del campo que desea contar. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto.

Aunque expr puede realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos null a menos que expr sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. Count(*) es considerablemente más rápida que Count(Campo). No se debe poner el asterisco entre dobles comillas ('*').

```
SELECT Count(*)AS Total FROM Pedidos;
```

Si expr identifica a múltiples campos, la función Count cuenta un registro sólo si al menos uno de los campos no es Null. Si todos los campos especificados son Null, no se cuenta el registro. Hay que separar los nombres de los campos con ampersand (&). `SELECT Count(FechaEnvío & Transporte)AS Total FROM Pedidos;`

5.20.4 Max, Min

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo específico de una consulta. Su sintaxis es:

Min(expr)Max(expr)

En donde expr es el campo sobre el que se desea realizar el cálculo. Expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
1 | SELECT Min(Gastos) AS ElMin FROM Pedidos WHERE Pais = 'España';
2 | SELECT Max(Gastos) AS ElMax FROM Pedidos WHERE Pais = 'España';
```

5.20.5 StDev, StDevP

Devuelve estimaciones de la desviación estándar para la población (el total de los registros de la tabla) o una muestra de la población representada (muestra aleatoria) . Su sintaxis es: **StDev(expr)StDevP(expr)**

En donde expr representa el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

StDevP evalúa una población, y StDev evalúa una muestra de la población. Si la consulta contiene menos de dos registros (o ningún registro para StDevP), estas funciones devuelven un valor Null (el cual indica que la desviación estándar no puede calcularse).

```
1 | SELECT StDev(Gastos) AS Desviacion FROM Pedidos WHERE Pais = 'España';
2 | SELECT StDevP(Gastos) AS Desviacion FROM Pedidos WHERE Pais = 'España';
```

5.20.6 Sum

Devuelve la suma del conjunto de valores contenido en un campo específico de una consulta. Su sintaxis es: **Sum(expr)**

En donde expr es el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
SELECT Sum(PrecioUnidad * Cantidad)AS Total FROM DetallePedido;
```

5.20.7 Var, VarP

Devuelve una estimación de la varianza de una población (sobre el total de los registros) o una muestra de la población (muestra aleatoria de registros) sobre los valores de un campo. Su sintaxis es:

Var(expr)VarP(expr)

VarP evalúa una población, y Var evalúa una muestra de la población. Expr el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

Si la consulta contiene menos de dos registros, Var y VarP devuelven Null (esto indica que la varianza no puede calcularse). Puede utilizar Var y VarP en una expresión de consulta o en una Instrucción SQL.

```
1 | SELECT Var(Gastos) AS Varianza FROM Pedidos WHERE Pais = 'España';
2 | SELECT VarP(Gastos) AS Varianza FROM Pedidos WHERE Pais = 'España';
```


Capítulo 6

Conceptos básicos

6.1 Introducción a la programación Web

En toda conexión web existen dos partes separadas: cliente y servidor¹. El cliente es la máquina del usuario que utiliza un navegador web y hace una petición al servidor; éste, recibe dicha petición, ya que en él reside el código de las diferentes páginas y la base de datos y es donde, en principio, se realiza el procesamiento.

6.1.1 Entorno

Escribimos en la barra de direcciones del navegador la url de la página web a ver. El navegador envía el mensaje a través de internet al servidor, según un protocolo estandarizado, solicitando la página (archivo) index.php.

El servidor recibe el mensaje, comprueba que se trata de una petición válida, si la extensión es "php" solicita al intérprete de PHP (programa que se ejecuta en el servidor web) que envíe el archivo.

En este caso la situación cambia: no es una simple extracción de un archivo desde el disco duro, sino que está actuando un agente intermediario: el intérprete PHP. Éste, lee desde el disco duro del servidor el archivo index.php y procesa el código de programación de dicho archivo. Decimos que el intérprete PHP "ejecuta" los comandos del archivo y, eventualmente, se comunica con un gestor de base de datos (como MySQL, MariaDB, Oracle, SQL Server, PostgreSQL, SQLite3, etc.). Cuando existe comunicación con una base de datos interviene otro agente más, el gestor de base de datos, que devuelve la información contenida en una base de datos.

Una vez que el intérprete PHP ejecuta el código del archivo y recibe la información necesaria del gestor de base de datos, envía los resultados al servidor. Éste, envía la página al cliente solicitante y el navegador muestra en pantalla la información que le envía el servidor web.

Una pregunta interesante ¿cuál es la diferencia entre el código HTML que recibe el cliente cuando solicita una página estática y el código HTML que recibe cuando solicita una página dinámica?. La respuesta es que no hay diferencia: ambos son código HTML. ¿Cómo sabemos si un código HTML proviene de un archivo html, estático, o si proviene de una respuesta de un intérprete PHP? Digamos que si nos dan el código no sabemos de dónde viene.

Y otra pregunta: si con PHP o sin él, obtenemos código HTML, ¿para qué sirve el PHP? PHP es un

¹Entendemos Servidor WEB

lenguaje de programación, mientras que HTML no lo es. Con HTML enviamos cierta información siempre igual, pero no hacemos cálculos, ni tomamos decisiones, no repetimos procesos cierto número de veces. PHP aporta la potencia de la programación de computadores a las páginas web.

Si queremos que una aplicación web ejecute un proceso de venta con una tarjeta de crédito. Una vez que el cliente elige el producto, número de unidades requeridas y lugar de envío, realizamos cálculos con PHP. Si el cliente requiere más productos, repite el proceso anterior; pero si presiona Salir, obtiene el importe total y los datos de su compra; introduce el número de su tarjeta de crédito y su clave personal.

Usando PHP consultamos la base de datos del banco, luego la tabla de clientes el número de tarjetas de crédito, clave y saldo disponible. Una vez hecha esta consulta, usando PHP mostramos al usuario un resultado según la situación producida, como:

- Los datos no son válidos. La compra no ha realizado.
- Los datos son válidos. La compra se ha realizado.

PHP es un lenguaje de programación y HTML, no. Es una gran diferencia. ¿Si PHP es más potente, por qué no prescindir de HTML? Cada uno cumple su función: HTML es un lenguaje para que de forma rápida se envíe y/o muestre información en un computador. En este sentido, es más rápido enviar o mostrar la información en formato HTML que de otra manera (como un archivo ejecutable en el lado del cliente). Por eso HTML es relevante dentro del mundo de internet: es un formato usado para el envío y/o mostrado de información. Cómo ésta se haya generado es otra pregunta.

Veamos el siguiente esquema de un proceso completo.



Figura 6.1: Modo de trabajo de una aplicación WEB

6.1.2 Ejercicios propuestos

Responde a las siguientes preguntas indicando verdadero o falso y justificando brevemente tu respuesta:

1. Todos los servidores web trabajan con PHP, es la única manera de conseguir que se muestre una página web en el computador cliente.

2. El HTML generado con PHP no es distinguible del código HTML generado manualmente, a no ser que contemos con información adicional.
3. HTML es un lenguaje de programación para decidir y repetir, pero carece de funcionalidades de acceso a bases de datos.
4. Para crear páginas web dinámicas, HTML es un lenguaje obsoleto. Para su creación usamos PHP u otro lenguaje de última generación.

6.1.3 Tecnologías

A continuación, mencionamos algunas de las tecnologías utilizadas en la programación WEB.

6.1.3.1 Framework

Estructura conceptual y tecnológica de soporte definido, con componentes de software concretos, como base para que un proyecto de informático sea organizado y desarrollado. Incluye soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para desarrollar y unir los diferentes componentes de un proyecto. Cuando nos referimos al "framework Django" hablamos de un conjunto de bibliotecas y programas para realizar aplicaciones web en Python de manera sencilla y elegante que utilizando sólo Python.

6.1.3.2 Navegador WEB

Una aplicación Web opera a través de Internet, interpretando la información de archivos y sitios web para que seamos capaces de leerla.

El navegador interpreta código HTML generalmente, en el que está escrita la página web y lo presenta en pantalla para que el usuario interactúe con su contenido y navegue hacia otros lugares de la red mediante enlaces o hipervínculos.

La funcionalidad básica de un navegador web es la visualización de texto, posiblemente con recursos multimedia incrustados. Estas páginas web pueden estar ubicados en la computadora donde está el usuario, pero también pueden estar en cualquier otro dispositivo que esté conectado a la computadora del usuario o a través de Internet. Además, tienen hipervínculos que enlazan una porción de texto o una imagen a otro documento, relacionado con el texto o la imagen.

El seguimiento de enlaces de una página a otra, ubicada en cualquier computadora conectada a la Internet, se llama navegación, de donde se origina el nombre navegador.

6.1.3.3 HTML

HTML es el lenguaje predominante para la elaboración de páginas web. Describe la estructura y el contenido de la página en forma de texto y lo complementa con objetos como imágenes, video y sonido.

El HTML se escribe en forma de etiquetas, rodeadas por corchetes angulares (<, >). Describe, hasta un cierto punto, la apariencia de un documento, y puede incluir un script (JavaScript), el cual afectar el comportamiento de los navegadores web y otros procesadores de HTML. Consideramos a un documento HTML como el resultado final que envía nuestra aplicación al navegador del usuario para que se visualice. HTML por tanto no es un lenguaje de programación sino una manera de estructurar la información.

6.1.3.4 CSS

CSS es un lenguaje usado para definir la presentación de un documento estructurado. El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que sirven de estándar para los agentes de usuario o navegadores. La idea detrás del desarrollo de CSS es separar la estructura de un documento de su presentación. Complementa por tanto lo definido en HTML, de manera que si se quiere modificar la apariencia de la web no haya que modificar el HTML sino sólo las hojas de estilo (CSS).

6.1.3.5 Javascript

Es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, tipeado y dinámico. Se utiliza en el lado del cliente, implementado como parte de un navegador web para mejoras en la interfaz de usuario y páginas web dinámicas, en bases de datos locales al navegador. JavaScript se diseñó con sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript no están relacionados, tienen semánticas y propósitos diferentes. Todos los navegadores modernos interpretan JavaScript integrado en las páginas web.

6.2 Instalando LAMP en un servidor Debian

LAMP es un grupo de programas de código abierto que son instalados en conjunto, con el objetivo de habilitar a un servidor como prestador de los servicios de páginas web dinámicas, así como los de aplicaciones web. De hecho, es el acrónimo que representa al sistema operativo Linux, con el servidor de aplicaciones Apache, una base de datos MySQL/MariaDB, y el contenido dinámico es procesado mediante PHP.

6.2.1 Prerrequisitos

Para continuar con el material de este libro, necesitas un servidor Debian, una cuenta de usuario diferente a la de superusuario (root), con los permisos para utilizar el comando su y un cortafuegos (firewall) básico. Lo configuras siguiendo nuestro texto para Debian.

6.2.2 Instalar Apache y actualizar el cortafuegos

El servidor web, Apache es uno de los servidores web más populares en el mundo. Se encuentra bien documentado y ha sido utilizado en buena parte de la historia de la web.

Instala Apache usando el administrador de paquetes de Debian, apt:

```
1 | apt update
2 | apt install apache2
```

Estas sentencias son ejecutadas con los privilegios de superusuario. Pregunta por la contraseña de tu cuenta regular para verificar permisos.

Una vez autenticada tu contraseña, apt informa cuáles paquetes se instalan y cuánto espacio en disco es requerido. Escribe Y, después Enter para continuar, así, la instalación procede.

6.2.3 Ajuste del cortafuegos para el tráfico web

Asumiendo que seguiste las instrucciones de configuración inicial del servidor y habilitaste el cortafuegos UFW; ahora, supervisa el tráfico HTTP y HTTPS. Para hacerlo, verifica que UFW tiene un perfil de aplicación para Apache mediante el comando:

```
1 ufw app list
2 Output
3 Available applications:
4 Apache
5 Apache Full
6 Apache Secure
7 OpenSSH
```

Si solicitas la información del perfil "Apache Full", ves el tráfico está habilitado para los puertos 80 y 443:

```
1 ufw app info "Apache Full"
2 Output
3 Profile: Apache Full
4 Title: Web Server (HTTP,HTTPS)
5 Description: Apache v2 is the next generation of the omnipresent
   Apache web
6 server.
7
8 Ports:
9 80,443/tcp
```

Para el tráfico de entrada HTTP y HTTPS para este perfil, Escribe:

```
ufw allow in "Apache Full"
```

Comprueba que todo es correcto visitando la dirección IP pública de tu servidor en un navegador:

```
http://your_server_ip
```

La página web predeterminada de Apache para Debian, con propósitos informativos y de prueba, se ve como:

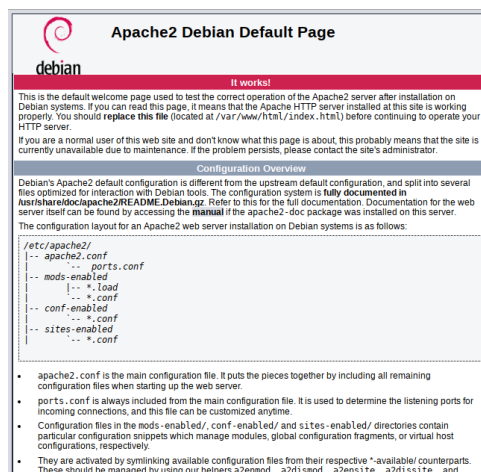


Figura 6.2: Información de la configuración del servidor

Si aparece esta página, tu servidor web está instalado correctamente y es accesible a través del cortafuegos.

Si no conoces la dirección IP pública de tu servidor, hay varias formas de encontrarla. Usualmente, es la dirección que usas para conectar tu servidor a través de SSH.

Hay diferentes formas de buscarla con la línea de comandos. En primer lugar, usa `iproute2` para obtener tu IP, escribe lo siguiente:

```
ip addr show eth0 | grep inet | awk 'print 2;' | sed 's/.*://' |
```

Esto regresa dos o tres líneas. Todas son direcciones correctas; sin embargo, es posible que tu computador sólo use una de ellas, recomendamos que las pruebes todas.

Un método alternativo es el uso de `curl` para contactar un servicio externo que informe cómo él ve a tu servidor. Esto se hace preguntando a un servidor específico cuál es tu dirección IP:

```
1 | apt install curl
2 | curl http://icanhazip.com
```

Sin importar el método para obtener tu dirección IP, escríbela en la barra de tu navegador web para ver la página predeterminada del Apache.

6.2.4 Instalar MySQL

Ahora que tienes tu servidor web activo y funcional, instala MariaDB o MySQL, para el acceso a las bases de datos de tu sitio.

Nuevamente, usa `apt` para instalar este software:

```
apt install mysql-server
```

Se despliega una lista de paquetes a instalar, así como el espacio en disco que es requerido. Presiona `Y` para continuar.

Cuando la instalación esté completa, ejecuta el archivo de comandos de seguridad que viene preinstalado en MySQL. El archivo remueve ciertos parámetros peligrosos y asegura el acceso a tu base de datos. Ejecuta el archivo así:

```
mysql_secure_installation
```

Pregunta si quieres configurar el conector de validación de contraseña: `VALIDATE PASSWORD PLUGIN`.

Nota: Esta funcionalidad depende de las necesidades del servidor. Si esta habilitada y una contraseña que no cumpla con un criterio específico, es rechazada por MySQL y genera error. Es un problema si utilizas contraseñas débiles en conjunto con software que configura credenciales de usuario MySQL, como los paquetes de Debian para phpMyAdmin. Deja esta validación deshabilitada; utiliza contraseñas únicas y fuertes para credenciales de las bases de datos.

Responde `Y` si estás de acuerdo, cualquier otra respuesta continua sin la habilitación.

```
1 | VALIDATE PASSWORD PLUGIN can be used to test passwords
2 | and improve security. It checks the strength of password
3 | and allows the users to set only those passwords which are
4 | secure enough. Would you like to setup VALIDATE PASSWORD plugin?
5 |
6 | Press y|Y for Yes, any other key for No:
```

Si respondiste "yes", selecciona el nivel de validación de contraseña. Ten en cuenta que si presionas 2, el nivel más fuerte, recibirás errores al intentar utilizar una contraseña que no contenga números, letras mayúsculas y minúsculas, así como caracteres especiales. Además, la contraseña no está basada en palabras comunes de un diccionario.

```
1  There are three levels of password validation policy:
2
3  LOW      Length >= 8
4  MEDIUM Length >= 8, numeric, mixed case, and special characters
5  STRONG  Length >= 8, numeric, mixed case, special characters and
        dictionary file
6
7  Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG: 1
8  Sin importar el nivel escogido para VALIDATE PASSWORD PLUGIN, tu
    servidor te solicitará, a continuación, seleccionar y confirmar la
    contraseña para el usuario root de MySQL. Ésta es una cuenta
    administrativa dentro MySQL con privilegios incrementados. Puede
    ser entendida de manera similar a la cuenta root del servidor
    mismo (Sin embargo, estarás configurando una cuenta específica
    para MySQL). Asegúrate de utilizar una contraseña fuerte y única,
    no debería dejarse en blanco.
9
10 Si habilitaste la validación de contraseña, se te mostrará qué tan
    fuerte es la contraseña para la cuenta root que acabas de
    introducir y tu servidor preguntará si quieres cambiarla. Si crees
    que es adecuado como está, Escribe N para seleccionar "no" en la
    línea de comandos:
11
12 Using existing password for root.
13
14 Estimated strength of the password: 100
15 Change the password for root ? ((Press y|Y for Yes, any other key for
    No) : n
```

Para las siguientes preguntas, presiona Y, así como Enter en cada sugerencia. Esto remueve los usuarios anónimos y la base de datos de prueba, deshabilita ingresos remotos del root, y carga estas nuevas reglas, de tal modo que MySQL respete inmediatamente los cambios que se acaban de hacer.

En este punto, tu sistema de bases de datos se encuentra configurado y sigue con la instalación de PHP.

6.2.5 Instalar PHP

PHP es el lenguaje que procesa código para desplegar contenido dinámico. Ejecuta archivos, accede a tus bases de datos MySQL para obtener información, y maneja la visualización del contenido procesado en tu servidor web.

Una vez más usamos el sistema apt para instalar PHP y lo configuramos para que se ejecute sobre el servidor Apache y se comunique con la base de datos MySQL:

```
apt install php libapache2-mod-php php-mysql
```

Esto instala PHP sin problemas.

Puedes modificar la forma en que Apache sirve archivos cuando un directorio es solicitado. Si un

usuario solicita un directorio del servidor, Apache primero busca un archivo llamado index.html. Si queremos que el servidor web dé prelación a los archivos PHP sobre cualquier otro archivo, ejecuta con privilegios de superusuario el siguiente comando para abrir el archivo dir.conf con el editor nano:

```
nano /etc/apache2/mods-enabled/dir.conf
```

Debes ver esto:

```
1 | /etc/apache2/mods-enabled/dir.conf
2 | <IfModule mod_dir.c>
3 | DirectoryIndex index.html index.cgi index.pl index.php index.xhtml
   | index.htm
4 | </IfModule>
```

Ahora, mueve el archivo de index.php a la primera posición después de DirectoryIndex, debe quedar así:

```
1 | /etc/apache2/mods-enabled/dir.conf
2 | <IfModule mod_dir.c>
3 | DirectoryIndex index.php index.html index.cgi index.pl index.xhtml
   | index.htm
4 | </IfModule>
```

Cuando termines, graba y cierra el archivo con Ctrl + X. Confirma los cambios presionando Y, luego Enter para verificar la grabación del archivo.

A continuación, reinicia el servidor Apache para que tus cambios sean reconocidos, mediante el comando:

```
systemctl restart apache2
```

También, verificas el estado del servicio apache2 con systemctl:

```
1 | sudo systemctl status apache2
2 | Sample Output
3 | apache2.service - LSB: Apache2 web server
4 | Loaded: loaded (/etc/init.d/apache2; bad; vendor preset: enabled)
5 | Drop-In: /lib/systemd/system/apache2.service.d
6 | apache2-systemd.conf
7 | Active: active (running) since Tue 2018-04-23 14:28:43 EDT; 45s ago
8 | Docs: man:systemd-sysv-generator(8)
9 | Process: 13581 ExecStop=/etc/init.d/apache2 stop (code=exited, status
   | =0/SUCCESS)
10 | Process: 13605 ExecStart=/etc/init.d/apache2 start (code=exited,
    | status=0/SUCCESS)
11 | Tasks: 6 (limit: 512)
12 | CGroup: /system.slice/apache2.service
13 | 13623 /usr/sbin/apache2 -k start
14 | 13626 /usr/sbin/apache2 -k start
15 | 13627 /usr/sbin/apache2 -k start
16 | 13628 /usr/sbin/apache2 -k start
17 | 13629 /usr/sbin/apache2 -k start
18 | 13630 /usr/sbin/apache2 -k start
```

Para ampliar la funcionalidad de PHP instala módulos adicionales. Para ver las opciones disponibles de módulos y librerías de PHP, envía los resultados de apt search a less, un paginador para navegar dentro de la salida de otro comando:


```
apt search php- less|
```

Usa las flechas para moverte hacia arriba y abajo, presiona Q para salir.

Obtienes una lista con los componentes a instalar. El sistema muestra una corta descripción de cada uno de ellos:

```
1 | bandwidthd-pgsql/bionic 2.0.1+cv20090917-10debian amd64
2 | Tracks usage of TCP/IP and builds html files with graphs
3 |
4 | bluefish/bionic 2.2.10-1 amd64
5 | advanced Gtk+ text editor for web and software development
6 |
7 | cacti/bionic 1.1.38+ds1-1 all
8 | web interface for graphing of monitoring systems
9 |
10 | ganglia-webfrontend/bionic 3.6.1-3 all
11 | cluster monitoring toolkit - web front-end
12 |
13 | golang-github-unkwon-cae-dev/bionic 0.0~git20160715.0.c6aac99-4 all
14 | PHP-like Compression and Archive Extensions in Go
15 |
16 | haserl/bionic 0.9.35-2 amd64
17 | CGI scripting program for embedded environments
18 |
19 | kdevelop-php-docs/bionic 5.2.1-1ubuntu2 all
20 | transitional package for kdevelop-php
21 |
22 | kdevelop-php-docs-l10n/bionic 5.2.1-1ubuntu2 all
23 | transitional package for kdevelop-php-l10n
24 | \dots
25 | :
```

Para indagar más sobre las funcionalidades de cada módulo, busca su descripción en la web, o solicita la descripción larga de cada paquete, para ello escribe:

```
apt show package_name
```

La salida es extensiva, con un campo en particular llamado "Description" que tiene una explicación acerca de la funcionalidad del módulo.

Para las funcionalidades del módulo php-cli, escribe:

```
apt show php-cli
```

Y ves algo como lo siguiente:

```
1 | Output
2 | \dots
3 | Description: command-line interpreter for the PHP scripting language
4 | (default)
5 | This package provides the /usr/bin/php command interpreter, useful
6 | for
7 | testing PHP scripts from a shell or performing general shell
8 | scripting tasks.
9 |
10 | .
11 | PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-
12 | used
```

```
8 | open source general-purpose scripting language that is especially
   suited
9 | for web development and can be embedded into HTML.
10 | .
11 | This package is a dependency package, which depends on Debian's
   default
12 | PHP version (currently 7.2).
13 | \dots
```

Si decides instalar algún paquete, usa el comando `apt install`, igual que lo has hecho para otro software.

Para instalar `php-cli`, ejecuta:

```
apt install php-cli
```

Si deseas instalar más de un módulo, separalos por un espacio, semejante a lo siguiente:

```
apt install package1 package2 ...
```

En este punto, LAMP está instalado y configurado.

Antes de hacer cambios o de desplegar una aplicación, es prudente hacer una prueba de la configuración de PHP, quizá haya alguna situación que merezca atención en este momento.

6.2.6 Evaluar el procesamiento de PHP sobre tu servidor web

Para evaluar si tu sistema está configurado de manera correcta para PHP, crea un archivo de comandos básico, llamado `info.php`. Éste, debe ser alojado en un directorio llamado "web root". En Debian, este directorio se encuentra localizado en `/var/www/html/`. Crea el archivo escribiendo:

```
nano /var/www/html/info.php
```

Esto crea un archivo en blanco. Introduce el siguiente código PHP válido, dentro del archivo de texto:

```
1 | info.php
2 | <?php
3 | phpinfo();
4 | ?>
```

Cuando termines, graba y cierra el archivo.

Ahora prueba si tu servidor web se encuentra habilitado para desplegar el contenido generado por este archivo PHP. Para hacerlo, visita una página web específica en tu navegador, necesitas tu dirección pública de nuevo.

Escribe:

```
http://your_server_ip/info.php
```

La página debe ser similar la siguiente:

6.2.7 Información predeterminada de PHP para Debian

Esta página provee información básica sobre tu servidor, desde la perspectiva de PHP. Es útil cuando necesites hacer algún tipo de seguimiento o para verificar que la configuración deseada ha sido aplicada de manera correcta.

Si pudiste ver esta página en tu navegador, PHP está trabajando según lo esperado.


PHP Version 5.6.30-0+deb8u1	
	
System	Linux debian8 4.16.0-041600-generic #201804012230 SMP Sun Apr 1 22:31:39 UTC 2018 x86_64
Build Date	Feb 8 2017 08:50:48
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/05-opcache.ini, /etc/php5/apache2/conf.d/10-mysqlnd.ini, /etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-curl.ini, /etc/php5/apache2/conf.d/20-gd.ini, /etc/php5/apache2/conf.d/20-openssl.ini, /etc/php5/apache2/conf.d/20-mcrypt.ini, /etc/php5/apache2/conf.d/20-mysql.ini, /etc/php5/apache2/conf.d/20-mysqli.ini, /etc/php5/apache2/conf.d/20-odbc.ini, /etc/php5/apache2/conf.d/20-pdo_mysql.ini, /etc/php5/apache2/conf.d/20-pdo_odbc.ini, /etc/php5/apache2/conf.d/20-pdo_pgsql.ini, /etc/php5/apache2/conf.d/20-pdo_sqlite.ini, /etc/php5/apache2/conf.d/20-pgsql.ini, /etc/php5/apache2/conf.d/20-readline.ini, /etc/php5/apache2/conf.d/20-sqlite3.ini
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226.NTS
PHP Extension Build	API20131226.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	enabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, mcrypt.*, mdecrypt.*

Figura 6.3:

Probablemente quieras remover este archivo después de la prueba, éste puede dar información sobre tu servidor a usuarios no autorizados. Para hacer esto, Escribe el siguiente comando:

```
rm /var/www/html/info.php
```

Siempre puedes crear este archivo de nuevo en caso que necesites acceder a esta información en otra oportunidad.

Capítulo 7

Fundamentos de programación con PHP

7.1 PHP como lenguaje de programación

1995 es el origen de PHP. Sin embargo, fue a partir del año 1999 con la publicación de PHP 4 cuando este lenguaje de programación tomó auge. En sus primeras etapas de desarrollo, PHP se denominaba también "Zend Engine", nombre que provenía de sus creadores: Zeev Suraski y Andi Gutmans. En 2004 fue lanzada la versión 5 de PHP basada en el nuevo motor Zend Engine 2.0. Esta nueva versión ofrecía mejoras para aplicaciones en servidores dedicados, mejor soporte para la programación orientada a objetos y una extensión completamente nueva para el uso de MySQL. La versión PHP 6 estuvo en formato de borrador durante largo tiempo, sin llegar a publicarse.

La llegada en 2016 de PHP 7 mejora el lenguaje como la performance del parser. La principal ventaja es la velocidad. El equipo de desarrollo mejoró el rendimiento. Además, ocupa menos memoria ya que su núcleo ha sido renovado.

En definitiva, una aplicación PHP 7 aumenta la velocidad de ejecución de código. Por ello, nuestro sitio web tiene un mejor posicionamiento en los buscadores que consideran el tiempo de carga, como es el caso de Google.

Si utilizamos aplicaciones desarrolladas con versiones anteriores de PHP en la versión de PHP 7, considere que la sintaxis del lenguaje de programación no ha sufrido grandes cambios. Algunas extensiones antiguas no son compatibles, como es el caso de MySQL, que se reemplaza por MySQLi.

Antes de usar el interprete de PHP 7, comprobamos los requisitos de la aplicación y su compatibilidad.

Las estadísticas indican que cerca del 60 % de los sitios Web en Internet utilizan PHP que es el lenguaje de programación que goza de gran popularidad y difusión. Las páginas web de Wikipedia o Yahoo se apoyan en este lenguaje, lo que demuestra su potencia, sirva para pequeñas páginas web hasta para grandes portales.

Muchas de las aplicaciones que se generan en comunidades de programadores y usuarios de software libre usan PHP porque está disponible gratuitamente y es potente.

¿Es PHP un buen lenguaje de programación? Para responder tomamos en cuenta muchos factores ya que el resultado final de un desarrollo de una aplicación web no sólo depende de la herramienta

sino de quién y cómo la maneja. Afirmamos que PHP utilizado correctamente es un muy buen lenguaje de programación.

Lo que está claro es que, aunque tiene ventajas e inconvenientes, es uno de los lenguajes de programación más difundidos en Internet y se encuentra disponible en prácticamente la totalidad de los servidores dedicados. Cuando algo tiene éxito y una amplia difusión, "por algo será".

PHP es quizás el lenguaje de programación más usado en el mundo y también es un lenguaje de programación orientado a objetos POO.

7.1.1 ¿Qué versión usar de PHP?

La versión de PHP a usar depende en gran medida del desarrollo a realizar. Si usamos PHP para el gestor de contenidos Joomla 1.0, no es lo mismo usarlo para 1.5 porque cada uno tiene sus propios requerimientos. Si una aplicación que fue concebida para PHP 4 y la intentamos ejecutar usando PHP 5, probablemente no trabaje bien.

Consideremos que usar la última versión tiene riesgos. Las últimas versiones no están probadas como las anteriores y pueden producir "agujeros de seguridad" que solventamos con algún parche. Usar la versión de PHP que mejor se adapte a nuestras necesidades y, dentro de ésta, la más estable.

7.1.2 Ejercicio propuesto

1. Busca información en internet sobre el actual estado de desarrollo de las versiones PHP. ¿Cuál es la última versión estable publicada? ¿En qué versiones se está trabajando como borrador?
2. Para comprobar si tus respuestas y código son correctos consulta los foros.

7.1.3 Ejecución de un programa PHP

PHP es un lenguaje diseñado para crear contenido HTML y puede ser ejecutado de tres formas:

1. en un servidor web,
2. desde de la línea de comandos,
3. mediante un cliente GUI.

El lenguaje puede ejecutarse en prácticamente todos los sistemas operativos actuales y en múltiples servidores web. PHP soporta una amplia variedad de bases de datos y cuenta con múltiples librerías para ejecutar procesos comunes.

Una aplicación PHP consiste de una página HTML con comandos PHP incrustados en ella. El servidor web procesa los comandos PHP y envía la salida al visualizador (browser).

El servidor web procesa código PHP y envía la salida al navegador. Un ejemplo de una página PHP sencilla es el siguiente:

```
1 <html>
2   <head> <title>Hello, world</title> </head>
3   <body>
```

Enteros
Flotantes
String
arreglos
Objetos
Variables variables

```
4  <?php echo "Hello , world!"; ?>
5  </body>
6  </html>
```

El comando echo de PHP inserta valores devueltos por la petición en la página HTML. Note que el código PHP se escribe dentro de los delimitadores `<?php` y `?>`.

Las sentencias PHP terminan con `;`, en el caso de ser la última instrucción no es necesario el punto y coma.

Los comentarios en PHP pueden ser: Como en C, `/* ... */` o `//`

Otro tipo de comentario es `#`, que comenta la línea en la que aparezca pero sólo hasta `?>` que cierra el código php.

7.1.4 Tipos de Datos

Los tipos de cada variable en PHP no están tan claros como en C. El intérprete asigna el tipo de una variable según el uso que se haga de ella. Para asignar un tipo fijo a una variable utiliza la función `settype()`. Los tipos son: El siguiente programa muestra la utilización de los tipos: datos enteros y flotantes.

```
1  <html>
2  <head> <title>Ejemplo 2 </title></head>
3  <body>
4    <h1> Ejemplo de PHP </h1>
5
6  <?php
7
8    #Enteros
9    $a = 1234; # número decimal
10   $a = -123; # un número negativo
11   $a = 0123; # número octal (equivalente al 83 decimal)
12   $a = 0x12; /* número hexadecimal (equivalente al 18 decimal) */
13
14   //Flotantes o reales
15   $b = 1.234; $b = 1.2e3;
16
17   //Escribimos algo
18   print "\n La a= $a y la b= $b <br>\n";
19 ?>
20
21 </body>
22 </html>
```

7.1.5 Cadenas de texto

Las cadenas de texto pueden estar delimitadas por `."`. Si la cadena de texto está delimitada por comillas dobles, cualquier variable dentro de ellas es sustituida por su valor (ejecutar el ejemplo anterior). Para especificar el carácter "se escapa con backslash(`\`).

Las operaciones con cadenas de texto son exactamente igual que en PERL. Con `strlen` se ve el tamaño de una cadena de texto y con el punto (`.`) se concatenan Cadenas de texto.

```
1 <html>
2 <head> <title>Ejemplo 3 </title></head>
3 <body>
4   <h1> Ejemplo de PHP </h1>
5
6 <?php
7   /* Asignando una cadena de texto. */
8   $str = "Esto es una cadena de texto";
9
10  /* Añadiendo a la cadena de texto. */
11  $str = $str . " con algo más de texto";
12
13  /* Otra forma de añadir, incluye un carácter de nueva línea */
14  $str .= " Y un carácter de nueva línea al final.\n";
15  print "$str <br>\n";
16
17  /* Esta cadena de texto terminará siendo '<p>Número: 9</p>' */
18  $num = 9;
19  $str = "<p>Número: $num</p>";
20  print "$str <br>\n";
21
22  /* Esta será '<p>Número: $num</p>' */
23  $num = 9;
24  $str = '<p>Número: $num</p>';
25  print "$str <br>\n";
26
27  /* Obtener el primer carácter de una cadena de texto como una
28     vector*/
29  $str = 'Esto es una prueba.';
30  $first = $str[0];
31  print "$str 0->$first <br>\n";
32
33  /* Obtener el último carácter de una cadena de texto. */
34  $str = 'Esto es aún una prueba.';
35  $last = $str[strlen($str)-1];
36  print "$str last->$last <br>\n";
37  ?>
38 </body>
39 </html>
```

Para convertir una cadena de texto a otro tipo de dato considera que una cadena de texto se evalúa como un valor numérico, cuyo valor resultante y tipo se determinan como sigue (El valor viene dado por la porción inicial de la cadena de texto. Cuando la primera expresión es una cadena de texto, el tipo de la variable depende de la segunda expresión.):

- Si la cadena de texto contiene cualquiera de los caracteres ".", "e", o "E". Se evalúa como un número doble, en caso contrario, como un entero.
- Si la cadena de texto comienza con datos de valor numérico, este es el valor usado. En caso contrario, el valor es 0 (cero).

```

1 <html>
2 <head> <title>Ejemplo 4</title></head>
3 <body>
4   <h1> Ejemplo de PHP </h1>
5
6 <?php
7
8   $foo = 1 + "10.5";           // $foo es doble (11.5)
9   print "$foo <br>\n";
10  $foo = 1 + "-1.3e3";         // $foo es doble (-1299)
11  print "$foo <br>\n";
12  $foo = 1 + "bob-1.3e3";      // $foo es entero (1)
13  print "$foo <br>\n";
14  $foo = 1 + "bob3";           // $foo es entero (1)
15  print "$foo <br>\n";
16  $foo = 1 + "10 Cerditos";    // $foo es entero (11)
17  print "$foo <br>\n";
18  $foo = 1 + "10 Cerditos";    // $foo es entero (11)
19  print "$foo <br>\n";
20  $foo = "10.0 cerdos " + 1;    // $foo es entero (11)
21  print "$foo <br>\n";
22  $foo = "10.0 cerdos " + 1.0;  // $foo es doble (11)
23  print "$foo <br>\n";
24
25 ?>
26 </body>
27 </html>

```

7.1.6 Arreglos

Los arreglos en PHP se utilizan tanto como arreglos indexados (vectores) o como asociativos (tablas hash). Para PHP, no existe diferencia entre arreglos indexados unidimensionales y los asociativos. Las funciones `list()` o `array()` son para crear arreglos. También se asigna el valor de cada elemento del array de manera explícita. En el caso de no especificar el índice en un array, el elemento asignado se añade al final.

```

1 <html>
2 <head> <title>Ejemplo 5</title></head>
3 <body>
4   <h1> Ejemplo de PHP </h1>
5
6 <?php
7
8   #forma explícita
9   $a[0] = "abc";

```



```

10  $a[1] = "def";
11  $b["foo"] = 13;
12
13  #Añadiendo valores al array
14  $a[] = "hola"; // $a[2] == "hola"
15  $a[] = "mundo"; // $a[3] == "mundo"
16
17  #mostramos los resultados
18  print "a= $a[0] , $a[1] , $a[2] , $a[3] <br>\n";
19  print "b[foo]=". $b["foo"]. "<br>\n";
20
21  ?>
22
23  </body>
24  </html>

```

Los arreglos se ordenan con las funciones `asort()`, `arsort()`, `ksort()`, `rsort()`, `sort()`, `uasort()`, `usort()`, y `uksort()` dependiendo del tipo de ordenación requerida.

El número de elementos en un array se cuenta con la función `count()`.

Un array se recorre usando las funciones `next()` y `prev()`. Otra forma de recorrerlo es con la función `each()`.

Los arreglos multidimensionales son simples, para cada dimensión array, se añade otro valor [clave] al final. Los índices de un array multidimensional pueden ser numéricos o asociativos.

```

1  $a[1]      = $f;           # ejemplos de una sola dimensión
2  $a["foo"]  = $f;
3
4  $a[1][0]   = $f;           # bidimensional
5  $a["foo"][2] = $f;         # (se pueden mezclar índices numéricos y
    asociativos)
6  $a[3]["bar"] = $f;         # (se pueden mezclar índices numéricos y
    asociativos)
7
8  $a["foo"][4]["bar"][0] = $f; # tetradimensional!

```

Los arreglos se declaran utilizando la instrucción `array` y se pueden llenar usando `=>`

```

1  # Ejemplo 1:
2  $a["color"]    = "rojo";
3  $a["sabor"]    = "dulce";
4  $a["forma"]    = "redondeada";
5  $a["nombre"]   = "manzana";
6  $a[3]          = 4;
7
8  # Ejemplo 2:
9  $a = array(
10     "color" => "rojo",
11     "sabor" => "dulce",
12     "forma" => "redondeada",
13     "nombre" => "manzana",
14     3       => 4
15 );

```

7.1.7 Objetos

Para inicializar un objeto utilizamos el método new y para acceder a sus métodos, el operador ->.

```
1 class nada {
2     function haz_nada () {
3         echo "No estoy haciendo nada.";
4     }
5 }
6
7 $miclase = new nada;
8 $miclase->haz_nada();
```

7.1.8 Conversión de Tipos de datos

Una variable en PHP, define su tipo según contenido y contexto en el que se utilice, es decir, si se asigna una cadena de texto a una variable, el tipo es string . Si a la misma variable se le asigna un número, el tipo cambia a entero.

Para configurar el tipo de una variable utiliza la función settype() y para obtener su tipo utiliza la función gettype().

También es posible utilizar el mecanismo casting como en lenguaje C.

```
1 <html>
2 <head> <title>Ejemplo 6</title></head>
3 <body>
4     <h1> Ejemplo de PHP </h1>
5
6 <?php
7
8     $foo = 10;    // $foo es un entero
9     $bar = (double) $foo;    // $bar es un doble
10
11     #Mostramos resultados
12     print "bar=$bar , foo=$foo <br>\n";
13
14 ?>
15
16 </body>
17 </html>
```

Los tipos de casting son:

- (int), (integer) - fuerza a entero (integer)
- (real), (double), (float) - fuerza a doble (double)
- (string) - fuerza a cadena de texto (string)
- (array) - fuerza a array (array)
- (object) - fuerza a objeto (object)

7.2 Variables

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

Los nombres de variables siguen las mismas reglas que otras etiquetas en PHP. Un nombre de variable valido tiene que empezar con una letra o una raya (underscore), seguido de cualquier número de letras, números y rayas. Como expresión regular se expresa como: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

Las variables se asignan por valor, pero desde PHP4, también se asignan por referencia usando el símbolo &.

```
1 <html>
2 <head> <title>Ejemplo 7</title></head>
3 <body>
4   <h1> Ejemplo de PHP </h1>
5
6 <?php
7   $foo = 'Bob';           // Asigna el valor 'Bob' a $foo
8   $bar = &$foo;           // Referencia $foo vía $bar.
9   $bar = "Mi nombre es $bar"; // Modifica $bar...
10  echo $foo." <br>\n";      // $foo también se modifica.
11  echo $bar." <br>\n";
12 ?>
13
14 </body>
15 </html>
```

Ten en cuenta que sólo las variables con nombre son asignadas por referencia.

7.2.1 Variables predefinidas

En PHP cada vez que se ejecuta un script, existen variables que se crean y que informan del entorno en el que se está ejecutando dicho script.

Para obtener una lista de estas variables predefinidas se utiliza la funcion `PHPinfo()`.

De estas variables, algunas se crean dependiendo del servidor que esté utilizando y otras son propias de PHP.

Para un servidor Apache, la lista de variables es: las variables creadas por PHP son:

Nota: La lista no es exhaustiva ni pretende serlo. Es una guía del tipo de variables predefinidas disponibles en un script PHP.

7.2.2 Ámbito de una variable

El ámbito de una variable en PHP es igual que en C o Perl tomando en cuenta los archivos incluidos al principio de cada programa.

La única diferencia se encuentra en las variables globales, que están expresamente definidas dentro de las funciones.

```
1 <html>
2 <head> <title>Ejemplo 8</title></head>
3 <body>
```

GATEWAY_INTERFACE:
 SERVER_NAME
 SERVER_SOFTWARE
 SERVER_PROTOCOL
 REQUEST_METHOD
 QUERY_STRING
 DOCUMENT_ROOT
 HTTP_ACCEPT
HTTP_ACCEPT_CHARSET
 HTTP_ENCODING
HTTP_ACCEPT_LANGUAGE
 HTTP_CONNECTION
 HTTP_HOST
 HTTP_REFERER
HTTP_USER_AGENT
 REMOTE_ADDR
 REMOTE_PORT
 SCRIPT_FILENAME
 SERVER_ADMIN
 SERVER_PORT
SERVER_SIGNATURE
 PATH_TRANSLATED
 SCRIPT_NAME
 REQUEST_URL

 argv
 argc
 PHP_SELF
HTTP_COOKIE_VARS
 HTTP_GET_VARS
 HTTP_POST_VARS

```

4  <h1> Ejemplo de PHP </h1>
5
6  <?php
7      $a = 1;
8      $b = 2;
9
10     Function Sum () {
11         global $a, $b;
12
13         $b = $a + $b;
14     }
15
16     Sum ();
17     echo $b;
18
19     ?>
20
21 </body>
22 </html>

```

7.2.3 Variables variables

PHP tiene un mecanismo para mantener variables con un nombre no fijo. Por ejemplo:

```

1  $a = "hola";
2  $$a = "mundo";

```

El ejemplo anterior define dos variables, una denominada \$a que contiene el valor "hola" y otra que se llama \$\$a que contiene el valor "mundo"

Para acceder al valor de una variable: `echo "$a ${$a}";`

La instrucción anterior genera la salida "hola mundo".

Cuando utilizas variables debes resolver la ambigüedad de utilizar arreglos de variables de este tipo. Por ejemplo:

```

1  $a[1] provoca una ambigüedad para el intérprete, puesto que no sabe
    si desea utilizar la variable denominada a[1] o utilizar la
    variables $$a indexándola en su primer valor. Para esto se utiliza
    una sintaxis especial que es *{{$a}[1]* según se desee una opción
    u otra.

```

7.2.4 Variables de los formularios HTML

En un formulario en HTML, después de ser enviado, dentro del ámbito PHP se crean automáticamente una variable por cada uno de los objetos que contiene el formulario.

Veamos:

```

1  <html>
2  <head> <title>Ejemplo 9</title></head>
3  <body>
4      <h1> Ejemplo de Formulario 1 </h1>
5
6      <p>

```

```

7 Dame tu nombre !!!
8
9 <form action="ej10.php" method="post">
10     Nombre: <input type="text" name="nombre">
11     <input type="submit">
12 </form>
13
14 </body>
15 </html>

```

Cuando es enviado, PHP crea la variable \$nombre que contiene el valor introducido en el campo Nombre del formulario.

```

1 <html>
2 <head> <title>Ejemplo 10</title></head>
3 <body>
4     <h1> Ejemplo de PHP </h1>
5
6     <?php
7         print "<h2>Hola $nombre </h2>\n";
8     ?>
9
10 </body>
11 </html>

```

PHP maneja arreglos en el contexto de variables de formularios, pero sólo en una dimensión. Puedes agrupar variables relacionadas o usar esta característica para recuperar valores de un campo select input múltiple:

```

1 <html>
2 <head> <title>Ejemplo 11</title></head>
3 <body>
4     <h1> Ejemplo de Formulario 2 </h1>
5
6     <form action="ej12.php" method="post">
7         Nombre: <input type="text" name="personal[name]">
8         E-mail: <input type="text" name="personal[email]">
9         Cerveza: <br>
10         <select multiple name="beer[]">
11             <option value="warthog">Warthog
12             <option value="guinness">Guinness
13             <option value="stuttgarter">Stuttgarter Schwabenbru
14         </select>
15         <input type="submit">
16     </form>
17 </body>
18 </html>
19
20 <html>
21 <head> <title>Ejemplo 12</title></head>
22 <body>
23     <h1> Ejemplo de PHP </h1>
24
25     <?php
26

```

<code>_FILE_:</code>	Archivo que se está procesando.
<code>_LINE_:</code>	Línea del Archivo que se está procesando
<code>_PHP_VERSION:</code>	Versión de PHP.
<code>PHP_OS:</code>	Sistema operativo del cliente.
<code>TRUE:</code>	Verdadero.
<code>FALSE:</code>	Falso.
<code>E_ERROR:</code>	Error sin recuperación.
<code>E_WARNING:</code>	Error recuperable.
<code>E_PARSE:</code>	Error no recuperable (sintaxis).
<code>E_NOTICE:</code>	Puede Tratarde de un error o no. Continúa ejecución.

```

27 | print "<h2>Hola $personal[name] , ";
28 | print "tu email es $personal[email] y ";
29 | print "te gusta la cerveza $beer[0] </h2>\n";
30 |
31 | ?>
32 |
33 | </body>
34 | </html>

```

Si `track_vars` está activada (configuración previa a la compilación), las variables enviadas con los métodos POST o GET también se encuentran en los arreglos asociativos globales `HTTP_POST_VARS` y `HTTP_GET_VARS`.

7.2.5 Constantes

Las constantes en PHP están definidas por la función `define()` [<php_manual_es.html#function.define>](#) y no pueden ser redefinidas con otro valor.

Además, existen una serie de variables predefinidas como: Todas las constantes que empiezan por `"E_"` se utilizan con la función `error_reporting()`.

```

1 | <html>
2 | <head> <title>Ejemplo 14</title></head>
3 | <body>
4 |   <h1> Ejemplo de PHP </h1>
5 |
6 |   <?php
7 |   define("CONSTANTE", "hello world.");
8 |   echo CONSTANTE;
9 |
10 |   ?>
11 | </body>
12 | </html>

```

7.2.6 Expresiones y operadores

En PHP una expresión es cualquier cosa que contenga un valor. Las expresiones más simples son las variables y constantes; otras más complicadas son las funciones, ya que cada función devuelve un valor al ser invocada, es decir, contiene un valor, por lo tanto, es una expresión.

Todas las expresiones en PHP son igual que en C. Los operadores abreviados, los incrementos,

Asociatividad	Operadores
Izquierda	,
Izquierda	or
Izquierda	xor
Izquierda	and
Derecha	print
Izquierda	= += -= *= /= .= %= &= = ^= = <= >=
Izquierda	?:
Izquierda	
Izquierda	&&
Izquierda	
Izquierda	^
Izquierda	&
No posee	== != ===
No posee	<= >=
Izquierda	>>
Izquierda	.
Izquierda	* / %
Derecha	! ~ ++ - (int) (double) (string) (array) (object) @
Derecha	[
No posee	new

etc, son iguales. Incluso existen otros operadores como el operador "." que concatena valores de variables, o el operador "===" denominado operador de identidad que devuelve verdadero si las expresiones a ambos lados del operador contienen el mismo valor y a la vez son del mismo tipo. Por último, el operador "@" para el control de errores. Veamos un ejemplo:

```

1 <?php
2 $res = @mysql_query("select nombre from clientes")
3 or die ("Error en la selección, '$php_errormsg'");
4 ?>

```

Utilizamos el operador en mysql_query y en el caso de dar un error, se graba el mensaje devuelto en una variable denominada php_errormsg. Ésta, contiene el mensaje de error de cada instrucción y si ocurre otro error posterior, se borra el valor con la nueva cadena de texto.

PHP mantiene los operadores "que sirven para ejecutar un comando del sistema tal y como hace la función system()<php_manual_es.html#function.system>

En PHP existen dos operadores and y dos operadores or que son: "and", "&&" y "or", "||" respectivamente, que se diferencian en el orden de cada uno.

La tabla que resume la precedencia de cada uno de los operadores es:

```

1 <html>
2 <head> <title>Ejemplo 15</title></head>
3 <body>
4   <h1> Ejemplo de PHP </h1>
5
6 <?php
7
8   function double($i) {

```



```

9      return $i*2;
10  }
11
12
13  $b = $a = 5;          /* asignar el valor cinco a las variables $a y
14  $b */
15  $c = $a++;            /* postincremento, asignar el valor original de
16  $a (5) a $c */
17  $e = $d = ++$b;       /* preincremento, asignar el valor incrementado
18  de $b (6) a
19  $d y a $e */
20
21  /* en este punto, tanto $d como $e son iguales a 6 */
22  $f = double($d++);    /* asignar el doble del valor de $d antes
23  del incremento, 2*6 = 12 a $f */
24  $g = double(++$e);    /* asignar el doble del valor de $e después
25  del incremento, 2*7 = 14 a $g */
26  $h = $g += 10;        /* primero, $g es incrementado en 10 y termina
27  valiendo 24.
28
29  después el valor de la asignación (24) se
30  asigna a $h,
31  y $h también acaba valiendo 24. */
32
33  #Operador de ejecución
34  $output = `ls -al`;
35  echo "<pre>$output</pre><br>";
36
37  echo "<h3>Postincremento</h3>";
38  $a = 5;
39  echo "Debería ser 5: " . $a++ . "<br>\n";
40  echo "Debería ser 6: " . $a . "<br>\n";
41
42  echo "<h3>Preincremento</h3>";
43  $a = 5;
44  echo "Debería ser 6: " . ++$a . "<br>\n";
45  echo "Debería ser 6: " . $a . "<br>\n";
46
47  echo "<h3>Postdecremento</h3>";
48  $a = 5;
49  echo "Debería ser 5: " . $a-- . "<br>\n";
50  echo "Debería ser 4: " . $a . "<br>\n";
51
52  echo "<h3>Predecremento</h3>";
53  $a = 5;
54  echo "Debería ser 4: " . --$a . "<br>\n";
55  echo "Debería ser 4: " . $a . "<br>\n";
56  ?>
57
58 </body>
59 </html>

```

7.2.7 Estructuras de Control

PHP ofrece una sintaxis alternativa para 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.

```
1 <html>
2 <head> <title>Ejemplo 16</title></head>
3 <body>
4   <h1> Ejemplo de PHP </h1>
5
6   <?php
7
8
9   $a=8;
10  $b=6;
11
12  // Primer if
13  if ($a > $b) {
14      print "a es mayor que b<br>";
15      $b = $a;
16  }
17
18  // if alternativo
19  if ($a > $b):
20      print "A es mayor que B<br>";
21  endif;
22
23  // Segundo if (con else y elseif )
24  if ($a > $b) {
25      print "a es mayor que b<br>";
26  } elseif ($a == $b) {
27      print "a es igual que b<br>";
28  } else {
29      print "b es mayor que a<br>";
30  }
31
32  // Segundo if alternativo
33  if ($a > $b):
34      print "A es mayor que B<br>";
35      print "...";
36  elseif ($a == $b):
37      print "A es igual a B<br>";
38      print "!!!";
39  else:
40      print "B es mayor que A<br>";
41  endif;
42  ?>
43
44 </body>
45 </html>
```

Veamos las estructuras de control de PHP en la lista siguiente:

1	Estructura	Alternativa
---	------------	-------------

```

2 | If, if else, if elseif  if: endif;
3 | while while: endwhile;
4 | for for: endfor;
5 | do.. while
6 | foreach(array as :math:`value) - foreach(a rray as `key=>$value)
7 | switch switch: endswitch;
8 | continue
9 | break
10 | require()(Necesitan estar dentro de tags PHP)
11 | include()(Necesitan estar dentro de tags PHP)

```

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

A diferencia de `include()`, `require()` siempre lee el archivo referenciado, incluso si la línea en que está no se ejecuta nunca.

Si requieres incluir condicionalmente un archivo usa `include()`. La instrucción `conditional` no afecta a `require()`. No obstante, si la línea donde aparece `require()` no se ejecuta, tampoco ejecuta el archivo referenciado.

De forma similar, las estructuras de ciclo no afectan la conducta de `require()`. Aunque el código contenido en el archivo referenciado está todavía sujeto al ciclo, el propio `require()` sólo ocurre una vez. Esto significa que no se puede poner una instrucción `require()` dentro de una estructura de ciclo y esperar que incluya el contenido de un archivo distinto en cada iteración. Para hacer esto, usa una instrucción `include()`. Así, `require()` reemplaza su llamada por el contenido del archivo que requiere, e incluye, incluye y evalúa el archivo especificado.

```

1 | <?php
2 |     print "Hola Mundo !<br>\n";
3 | ?>

```

El archivo que realiza la inclusión del primero es similar a esto:

```

1 | <html>
2 | <head> <title>Ejemplo 18</title></head>
3 | <body>
4 |     <h1> Ejemplo de PHP </h1>
5 |
6 | <?php include( 'ej17.php' ); ?>
7 |
8 | </body>
9 | </html>

```

`require()` y `include()` son idénticas en todos los aspectos excepto en el modo de actuar ante un error.

7.2.7.1 `require()`

La sentencia `require()` incluye y evalúa el archivo especificado.

`include()` produce Warning¹ mientras que `require()` Error Fatal. En otras palabras, no dude en utilizar `require()` si quiere que un archivo no encontrado cuelgue el procesamiento de la página. `include()` no se comporta de esta manera, el script sigue funcionando de todas maneras. Asegurarse que `include_path` esté bien configurado.

¹Traducido como precaución o atención

7.2.7.2 include()

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

Esta documentación también se aplica a la función require().

Cuando un archivo es incluido, el código hereda la variable scope de la línea en donde el include ocurre.

Cualquier variable disponible en esa línea en el archivo desde donde se hace la inclusión está disponible en el archivo incluido a partir de ese momento.

7.3 Funciones

7.3.1 Funciones definidas por el usuario

Un ejemplo puede ser:

```
1 function foo($arg1, $arg2, ..., $argN) {
2     echo "Función ejemplo";
3     return $value;
4 }
```

Dentro de una función puede aparecer cualquier cosa, incluso otra función o definiciones de clase.

Respecto al paso de argumentos, son siempre pasados por valor y para pasarlos por referencia hay que indicarlo y se hace de dos formas diferentes, en la definición de la función, anteponiendo el símbolo & al argumento que corresponda, en este caso la llamada es igual que invocar una función normal, o manteniendo la definición de la función normal y anteponer un & delante del argumento que corresponda en la llamada a la función.

```
1 <html>
2 <head> <title>Ejemplo 19</title></head>
3 <body>
4   <h1> Ejemplo de PHP </h1>
5
6   <?php
7
8   //Define la función con parametros por referencia
9   function suma1 (&$a, &$b) {
10       $c=$a+$b;
11       return $c;
12   }
13
14   //Define la función con parametros por valor
15   function suma2 ($a, $b) {
16       $c=$a+$b;
17       return $c;
18   }
19
20   $a=2; $b=3; $suma;
21
22   //Llama la función 1 por referencia (no puede ser de otra forma)
23   print $suma=suma1($a,$b);
24
25   //Llama la función 2 por referencia
```

```

26 | print $suma=suma1(&$a,&$b);
27 |
28 | //Llama la función 2 por valor
29 | print $suma=suma1($a,$b);
30 |
31 | ?>
32 |
33 | </body>
34 | </html>

```

PHP facilita el mecanismo de argumentos por defecto. Un ejemplo de esta característica es:

```

1 | function hacerCafe($tipo="capuchino") {
2 |     return "he hecho un café $tipo\n";
3 | }

```

En la llamada a esta función se obtiene una frase u otra según se llame:

```

1 | echo hacerCafe();
2 | echo hacerCafe("expreso");

```

En el caso de una función con argumentos por defecto y argumentos normales, los argumentos por defecto están agrupados al final de la lista de argumentos.

En PHP4 el número de argumentos de una función definida por el usuario, puede ser variable, se utilizan las funciones `func_num_args()`, `func_get_arg()` y `func_get_args()`.

7.3.2 Valores devueltos

A diferencia de C, PHP puede devolver cualquier número de valores, sólo hace falta recibir estos argumentos de la forma adecuada. Ejemplo:

```

1 | function numeros() {
2 |     return array(0,1,2);
3 | }
4 |
5 | list ($cero, $uno, $dos) = numeros();

```

7.3.3 Funciones variables

PHP soporta el concepto de funciones variable, esto significa que si una variable tiene unos paréntesis añadidos al final, PHP busca una función con el mismo nombre que la evaluación de la variable, e intenta ejecutarla.

```

1 | <?php
2 | function foo() {
3 |     echo "En foo()<br\>\n";\
4 | }
5 |
6 | function bar ($arg='') {
7 |     echo " bar();El argumento ha sido '$arg'.<br\>\n";\
8 | }
9 |
10 | $func = 'foo';
11 | $func();

```

```

12 | $func='bar';
13 | $func('test');
14 | ?>

```

7.4 Procesamiento de formularios

Es sencillo procesar formularios con PHP, ya que los parámetros del formulario están disponibles en los arreglos `_GET` y `_POST`.

7.4.1 Métodos

Existen dos métodos HTTP que un cliente utiliza para pasar los datos del formulario al servidor: GET y POST. El método que utiliza un formulario particular, se especifica con el atributo `method` en la etiqueta `form`. En teoría, los métodos son sensibles a mayúsculas en el código HTML, pero en la práctica algunos navegadores fallan si el nombre del método no está en mayúsculas.

A solicitud, GET codifica los parámetros del formulario en la dirección URL en lo que se llama una cadena de consulta, el texto que sigue al carácter `?` es la cadena de consulta:

```
/path/to/page.php?keyword=bell&length=3
```

Una solicitud POST pasa los parámetros del formulario en el cuerpo de la solicitud HTTP, dejando intacta la URL. El tipo de método utilizado para solicitar una página PHP está disponible a través de `$_SERVER['REQUEST_METHOD']`. Por ejemplo:

```

1 | if ($_SERVER['REQUEST_METHOD'] == 'GET') {
2 |     // handle a GET request
3 | } else {
4 |     die("You may only GET this page.");
5 | }

```

7.5 Parámetros

Se utilizan los arreglos `_POST`, `_GET` y `_FILES` para acceder a los parámetros de formulario desde el código PHP. Las llaves son los nombres de los parámetros y los valores son los valores de esos parámetros. Considere la siguiente página utilizada para separar una palabra:

```

1 | <html>
2 | <head><title>Formulario</title></head>
3 | <body>
4 |     <form action="separar.php" method="POST">
5 |         Ingrese una palabra:
6 |         <input type="text" name="word"/><br/>
7 |         Largo de las separaciones:
8 |         <input type="text" name="number" /><br/>
9 |         <input type="submit" value="Dividir">
10 |     </form>
11 | </body>
12 | </html>

```

El programa PHP para procesar dicho formulario es el siguiente:

```

1 $word = $_POST['word'];
2 $number = $_POST['number'];
3 $chunks = ceil(strlen($word) / $number);
4 echo "The {$number}-letter chunks of '{$word}' are:<br />\n";
5 for ($i = 0; $i < $chunks; $i++) {
6     $chunk = substr($word, $i * $number, $number);
7     printf("%d: %s<br />\n", $i + 1, $chunk);
8 }

```

7.6 Páginas con auto-procesamiento

Una página PHP puede ser utilizada tanto para generar un formulario como para procesarlo.

```

1 <html>
2 <head><title>Temperature Conversion</title></head>
3 <body>
4     <?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
5         <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="
6             POST">
7             Fahrenheit temperature:
8             <input type="text" name="fahrenheit" /><br/>
9             <input type="submit" value="Convert to Celsius!" />
10        </form>
11        <?php } else if ($_SERVER['REQUEST_METHOD'] == 'POST') {
12            $fahrenheit = $_POST['fahrenheit'];
13            $celsius = ($fahrenheit - 32) * 5 / 9;
14            printf("%.2fF is %.2fC", $fahrenheit, $celsius);
15        } else {
16            die("This script only works with GET and POST requests.")
17            ;
18        } ?>
19    </body>
20 </html>

```

Otra forma de programa decide si mostrar un formulario o proceso es ver si alguno de los parámetros es suministrado. Esto escribe una página de auto-procesamiento que utiliza el método GET para enviar valores.

```

1 <html>
2 <head><title>Temperature Conversion</title></head>
3 <body>
4     <?php $fahrenheit = $_GET['fahrenheit'];
5     if (is_null($fahrenheit)) { ?>
6         <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="GET">
7             Fahrenheit temperature:
8             <input type="text" name="fahrenheit" /><br />
9             <input type="submit" value="Convert to Celsius!" />
10        </form>
11        <?php } else {
12            $celsius = ($fahrenheit - 32) * 5 / 9;
13            printf("%.2fF is %.2fC", $fahrenheit, $celsius); } ?>
14    </body>
15 </html>

```

7.7 Formularios adhesivos

Muchos sitios web utilizan una técnica conocida como formularios adhesivos, en el que los resultados de una consulta se acompañan de un formulario de búsqueda cuyos valores por defecto son los de la consulta anterior.

La técnica básica consiste en utilizar el valor enviado por el formulario como valor por defecto cuando se crea el campo HTML.

```
1 <html>
2 <head><title>Temperature Conversion</title></head>
3 <body>
4 <?php $fahrenheit = $_GET['fahrenheit']; ?>
5 <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
6     Fahrenheit temperature:
7     <input type="text" name="fahrenheit" value="<?php echo
        $fahrenheit; ?>" /><br/>
8     <input type="submit" value="Convert to Celsius!" />
9 </form>
10 <?php if (!is_null($fahrenheit)) {
11     $celsius = ($fahrenheit - 32) * 5 / 9;
12     printf("%.2fF is %.2fC", $fahrenheit, $celsius);
13 } ?>
14 </body>
15 </html>
```

7.8 Parámetros multivaluados

Las listas de selección HTML, creadas con la etiqueta select, permiten selecciones múltiples. Para asegurarse que PHP reconoce los múltiples valores que el navegador pasa a un programa de procesamiento de formularios, es necesario hacer que el nombre del campo en la formulario HTML finalice con [].

```
1 <html>
2 <head><title>Personality</title></head>
3 <body>
4 <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
5     Select your personality attributes: <br/>
6     <select name="attributes[]" multiple>
7         <option value="perky">Perky</option>
8         <option value="morose">Morose</option>
9         <option value="thinking">Thinking</option>
10        <option value="feeling">Feeling</option>
11        <option value="thrifty">Spend-thrift</option>
12        <option value="shopper">Shopper</option>
13    </select><br/>
14    <input type="submit" name="s" value="Record my personality!" />
15 </form>
16 <?php if (array_key_exists('s', $_GET)) {
17     $description = join(' ', $_GET['attributes']);
18     echo "You have a {$description} personality.";
19 } ?>
```



```

20     </body>
21 </html>

```

Otro ejemplo similar pero que utiliza checkbox es:

```

1 <html>
2   <head><title>Personality</title></head>
3   <body>
4     <form action="<?php $_SERVER['PHP_SELF']; ?>" method="GET">
5       Select your personality attributes:<br />
6       <input type="checkbox" name="attributes[]" value="perky" />
          Perky<br />
7       <input type="checkbox" name="attributes[]" value="morose" />
          Morose<br />
8       <input type="checkbox" name="attributes[]" value="thinking" />
          Thinking<br />
9       <input type="checkbox" name="attributes[]" value="feeling" />
          Feeling<br />
10      <input type="checkbox" name="attributes[]" value="thrifty" />
          Spend-thrift<br />
11      <input type="checkbox" name="attributes[]" value="shopper" />
          Shopper<br /><br />
12      <input type="submit" name="s" value="Record my personality!" />
13    </form>
14    <?php if (array_key_exists('s', $_GET)) {
15      $description = join(' ', $_GET['attributes']);
16      echo "You have a {$description} personality.";
17    } ?>
18  </body>
19 </html>

```

7.9 Parámetros multivaluados adhesivos

Para manejar parámetros multivaluados adhesivos es útil escribir una función para generar el código HTML de los valores posibles y trabajar a partir de una copia de los parámetros enviados.

```

1 <html>
2   <head><title>Personality</title></head>
3   <body>
4
5   <?php
6     $attrs = $_GET['attributes'];
7     if (!is_array($attrs)) {
8       $attrs = array();
9     }
10
11   function makeCheckboxes($name, $query, $options) {
12     foreach ($options as $value => $label) {
13       $checked = in_array($value, $query) ? "checked" : '';
14       echo "<input type=\"checkbox\" name=\"{$name}\""
15         . "value=\"{$value}\" {$checked} />";
16       echo "{$label}<br />\n";
17     }

```

```

18
19 $personalityAttributes = array(
20     'perky'=> "Perky",
21     'morose'=> "Morose",
22     'thinking'=> "Thinking",
23     'feeling'=> "Feeling",
24     'thrifty'=> "Spend-thrift",
25     'prodigal'=> "Shopper"
26 ); ?>
27
28 <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
29     Select your personality attributes:<br />
30     <?php makeCheckboxes('attributes', $attrs, $personalityAttributes
31         ); ?><br />
32     <input type="submit" name="s" value="Record my personality!" />
33 </form>
34
35 <?php if (array_key_exists('s', $_GET)) {
36     $description = join(' ', $_GET['attributes']);
37     echo "You have a {$description} personality.";
38 } ?>
39
40 </body>
41 </html>

```

7.10 Manteniendo el estado

HTTP es un protocolo sin estado, lo que significa que cada vez que un servidor web completa la petición de un cliente para una página web, la conexión entre los dos desaparece. En otras palabras, no hay manera en que un servidor pueda reconocer que toda secuencia de peticiones se originan desde el mismo cliente.

Para solucionar esta falta de estado del protocolo HTTP, existen varias técnicas para el seguimiento de la información de estado entre las solicitudes (conocido como el seguimiento de sesión).

7.10.1 Campos ocultos en formularios

Una de estas técnicas es el uso de campos ocultos de formulario para pasar la información. PHP trata a los campos de formulario ocultos al igual que los campos de formulario normales, por lo que los valores están disponibles en los arreglos `_GET` y `_POST`. Mediante el uso de campos de formulario ocultos, se puede mantener toda la información necesaria para cualquier aplicación. Sin embargo, una técnica común consiste en asignar a cada usuario un identificador único y pasar el identificador usando un único campo de formulario oculto. Mientras que los campos de formulario ocultos funcionan en todos los navegadores, ellos trabajan sólo para una secuencia de formularios generados de forma dinámica, por lo que generalmente no son tan útiles como algunas otras técnicas.

```

1 <?php
2     $number = $_POST['number'];
3     if (isset($number)) {
4         $count = intval($_POST['count']);

```

```

5     $count++;
6     $numbers = Array();
7     array_push($numbers,$number);
8     for ($i = 0; $i < $count-1; $i++) {
9         array_push($numbers,$_POST['number'.'. $i]);
10    }
11 } else {
12     $count = 0;
13 }
14 ?>
15 <html>
16     <head>
17         <title>My Lottery</title>
18     </head>
19     <body>
20 <h2>My Lottery</h2>
21 <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
22 <?php
23     if ($count == 0) {
24         echo "<h3>Wellcome!!</h3>";
25     } else {
26         echo "<label>Your winning numbers are: </label>";
27         for ($i = 0; $i < $count-1; $i++)
28             echo "<b>". $numbers[$i]. "</b>, ";
29         echo "<b>". $numbers[$count-1]. "</b></p>";
30     }
31     if ($count == 6) {
32         echo "<h3>Good luck!!</h3>";
33     } else {
34         ?>
35 <label>Please, enter a number:</label>
36 <input type='text' name='number' />
37 <input type='submit'>
38 <?php } ?>
39 <input type="hidden" value="<?php echo $count; ?>" name="count"/>
40 <?php
41     for ($i = 0; $i < $count; $i++) { ?>
42         <input type="hidden" value="<?php echo $numbers[$i]; ?>"
43             name="number<?php echo $i?>" />
44     <?php } ?>
45 </form>
46 </body>
47 </html>

```

7.11 Cookies

La segunda técnica de mantener el estado es el uso de cookies. Una cookie es un fragmento de información que el servidor asigna a un cliente. En cada solicitud posterior, el cliente devuelve esa información al servidor, identificándose así a sí mismo. Las cookies son útiles para conservar la información a través de las repetidas visitas de un navegador, pero también tienen sus propios problemas. El principal es que la mayoría de los navegadores dejan a los usuarios desactivar las

cookies. Por lo que cualquier aplicación que utiliza las cookies para el mantenimiento del estado tiene que usar otra técnica como un mecanismo de reserva.

Una cookie es básicamente una cadena que contiene varios campos. Un servidor puede enviar una o más cookies a un navegador en las cabeceras de una respuesta. Algunos de los campos de la cookie indican las páginas para las que el navegador debe enviar la cookie como parte de la solicitud.

PHP soporta cookies HTTP de forma transparente. Se configuran usando la función `setcookie()` o `setrawcookie()`. Las cookies son parte del header HTTP, así es que `setcookie()` debe ser llamada antes que cualquier otra salida sea enviada al browser. Los envíos de cookies desde el cliente son incluidos automáticamente en el arreglo:

```
1  $_COOKIE.
2
3  <?php
4      $number = $_POST['number'];
5      if (isset($number)) {
6          $count = intval($_COOKIE['count']);
7          setcookie('number' . $count, $number);
8          $count++;
9      } else {
10         foreach ($_COOKIE as $key => $value )
11             setcookie($key, FALSE);
12         $count = 0;
13     }
14     setcookie('count', $count);
15 ?>
16 <html>
17     <head>
18         <title>My Lottery</title>
19     </head>
20     <body>
21         <h2>My Lottery</h2>
22         <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="
23             POST">
24             <?php
25                 if ($count == 0) {
26                     echo "<h3>Wellcome!!</h3>";
27                 } else {
28                     echo "<label>Your winning numbers are: </label>";
29                     for ($i = 0; $i < $count-1; $i++)
30                         echo "<b>".$_COOKIE['number' . $i]."</b>, ";
31                     echo "<b>$number</b></p>";
32                 }
33                 if ($count == 6) {
34                     echo "<h3>Good luck!!</h3>";
35                 } else {
36                     ?>
37                     <label>Please, enter a number:</label>
38                     <input type='text' name='number' />
39                     <input type='submit'>
40                     <?php } ?>
41                 </form>
```

```
41     </body>
42 </html>
```

7.11.1 Uso de sesiones

La mejor manera de mantener el estado con PHP es utilizar el sistema integrado de seguimiento de sesiones. Este sistema crea variables persistentes que son accesibles desde diferentes páginas de la aplicación, así como en diferentes visitas al sitio por el mismo usuario. Internamente, el mecanismo de seguimiento de la sesión de PHP utiliza cookies (o URLs) para resolver con elegancia la mayoría de los problemas que requieren del estado, cuidando de todos los detalles para el programador.

La función PHP `session_start()` crea una sesión o reanuda la actual basada en un identificador de sesión pasado mediante una petición GET o POST, o pasado mediante una cookie. El soporte para sesiones almacena los datos entre peticiones en el arreglo `$_SESSION`. La función `session_destroy()` destruye toda la información asociada con la sesión actual. No destruye las variables globales asociadas con la sesión, ni destruye la cookie de sesión.

```
1  <?php
2      session_start();
3      $number = $_POST['number'];
4      if (isset($number)) {
5          $count = intval($_SESSION['count']);
6          $_SESSION['number' . $count] = $number;
7          $count++;
8      } else {
9          session_destroy();
10         $count = 0;
11     }
12     $_SESSION['count'] = $count;
13 ?>
14 <html>
15     <head>
16         <title>My Lottery</title>
17     </head>
18     <body>
19         <h2>My Lottery</h2>
20         <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="
21             POST">
22             <?php
23                 if ($count == 0) {
24                     echo "<h3>Wellcome!!</h3>";
25                 } else {
26                     echo "<label>Your winning numbers are: </label>";
27                     for ($i = 0; $i < $count-1; $i++)
28                         echo "<b>".$_SESSION['number' . $i]."</b> , ";
29                     echo "<b>$number</b></p>";
30                 }
31                 if ($count == 6) {
32                     echo "<h3>Good luck!!</h3>";
33                 } else {
```

```

34     <label>Please , enter a number:</label>
35     <input type='text' name='number' />
36     <input type='submit'>
37     <?php } ?>
38     </form>
39 </body>
40 </html>

```

7.11.2 Reescritura del URL

Otra técnica es la reescritura de URL, donde cada URL local en la que el usuario puede hacer clic se modifica dinámicamente para incluir información adicional. Esta información adicional se especifica como un parámetro en la URL. Por ejemplo, si se asigna a cada usuario un identificador único, es posible incluir ese ID en todas las direcciones URL, de la siguiente manera:

`http://www.example.com/catalog.php?userid=123`

Si es posible modificar dinámicamente todos los enlaces locales para incluir un ID de usuario, se podrá realizar un seguimiento de los usuarios individuales en su aplicación. La reescritura de URL trabaja para todos los documentos generados dinámicamente, y no sólo los formularios, pero en realidad llevar a cabo la reescritura puede ser tedioso.

7.11.3 Ejemplo

A continuación se muestra una aplicación Web en PHP que lleva una lista de eventos. Cada evento tiene una fecha, hora, y asunto. Se pueden agregar nuevos eventos o eliminar los existentes. En este caso se utilizan "cookies" para resolver el problema. Note como se crea un cadena de texto (string) con los diferentes campos a almacenar utilizando como delimitador el carácter "|", y luego se genera otra cadena de texto con todos los eventos del arreglo pero ahora delimitados por los caracteres "/n". Esta última cadena de texto es la que se almacena en la "cookie".

```

1  <?php
2      if (isset($_COOKIE['eventos'])) {
3          $array = explode("/n", $_COOKIE['eventos']);
4      } else {
5          $array = array();
6      };
7
8      if (isset($_POST['borrar'])) {
9          $id = $_POST['borrar'];
10         unset($array[$id]);
11         $array = array_values($array);
12         setcookie('eventos', implode("/n", $array));
13     } else if (isset($_POST['agregar'])) {
14         $new_item = $_POST['dia'].'|'.$_POST['hora'].'|'.$_POST['evento'];
15         $array[] = $new_item;
16         setcookie('eventos', implode("/n", $array));
17     } else {
18         setcookie('eventos', null);
19         $array = array();
20     }

```

```

21  ?>
22  <html>
23      <meta http-equiv="Content-Type" content="text/html; charset=UTF
      -8" />
24      <head>
25          <title>Calendario de eventos</title>
26      </head>
27      <body>
28          <h2>Calendario de eventos</h2>
29          <form action="php echo $_SERVER['PHP_SELF']; ?" method="POST
      ">
30              <table border=1>
31                  <tr><th>Día</th><th>Hora</th><th>Evento</th><th>Operación
      </th></tr>
32                  <?php
33                      for ($i=0;$i<sizeof($array);$i++) {
34                          $values = explode("|", $array[$i]);
35                          echo '<tr><td>'.$values[0].'</td><td>'.$values[1].'</
      td><td>'.
36                              $values[2].'</td><td><button name="borrar" value
      ="'.'. $i.
37                              '>Borrar</button></td></tr>';
38                      }
39                  <?>
40                  <tr><td><input size="10" name="dia" type="date"/></td>
41                      <td><input name="hora" size="10" type="time"/></td>
42                      <td><input size="40" name="evento"/></td>
43                      <td><button name="agregar">Agregar</button></td>
44                  </tr>
45              </table>
46          </form>
47      </body>
48  </html>

```

7.11.4 Ejercicio propuesto

1. Modifique el caso anterior para manejar los eventos mediante campos ocultos.
2. Modifique el mismo ejemplo para almacenar los objetos mediante variables de sesión.

7.12 Manejo de archivos

La función `fopen(path,mode)` abre un archivo local o mediante un URL. El `path` del archivo debe incluir la ruta completa al mismo. El `mode` puede ser `r` - lectura, `w` - escritura, `a` - agregar, o `x` - escritura exclusiva. Se puede agregar un `+` al modo y si el archivo no existe, se intenta crear. La función `fclose(file)` cierra un archivo abierto.

La función `feof(file)` comprueba si el puntero a un archivo se encuentra al final del archivo. La función `fgets(file)` obtiene una línea desde el puntero a un archivo. La función `file_exists(file)` comprueba si existe un archivo o directorio.

```
1  <?php
```

```

2
3 $path = "/home/user/file.txt";
4 if (!file_exists($path))
5     exit("File not found");
6 $file = fopen($path, "r");
7 if ($file) {
8     while (($line = fgets($file)) !== false) {
9         echo $line;
10    }
11    if (!feof($file)) {
12        echo "Error: EOF not found\n";
13    }
14    fclose($file);
15 }
16
17 ?>

```

La función `fscanf` analiza la entrada desde un archivo de acuerdo a un formato. Los tipos más importantes son: `%d` - entero, `%f` - flotante, y `%s` - string. Un detalle importante es que `%s` no reconoce cadenas de texto con espacios en blanco, únicamente palabras completas.

```

1 <?php
2
3 $path = "/home/usr/data.txt";
4 if (!file_exists($path))
5     exit("File not found");
6 $file = fopen($path, "r");
7 echo "<html><body><table border=1>";
8 echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density
   </th></tr>";
9 while ($data = fscanf($file, "%s\t%d\t%d\t%f\n")) {
10     list ($country, $area, $pop, $dens) = $data;
11     echo "<tr><td>". $country. "</td><td>". $area. "</td><td>".
12         $pop. "</td><td>". $dens. "</td></tr>";
13 }
14 echo "</table></body></html>";
15 fclose($file);
16
17 ?>

```

El archivo de datos para el ejemplo anterior podría ser el siguiente. Note que debe haber un tabulador que separe cada campo de un mismo registro.

Belice	22966	334000	14.54
Costa_Rica	51100	4726000	92.49
El_Salvador	21041	6108000	290.29
Guatemala	108894	15284000	140.36
Honduras	112492	8447000	75.09
Nicaragua	129494	6028000	46.55
Panama	78200	3652000	46.70

7.12.1 Directorios

La función `is_dir` indica si el nombre del archivo es un directorio y la función `is_file` indica si el nombre de archivo es un archivo. La función `mkdir` crea un directorio. La función `rename` renombra un archivo o directorio. La función `rmdir` remueve un directorio y la función `unlink` remueve un archivo.

7.12.2 Archivos binarios

Las funciones `fread` y `fwrite` leen y escriben, respectivamente, un archivo en modo binario. La función `fseek` posiciona el puntero del archivo.

```
1  <?php
2
3  $path = "/home/usr/data2.txt";
4  if (!file_exists($path))
5      exit("File not found");
6  $file = fopen($path, "r");
7  echo "<html><body><table border=1>";
8  echo "<tr><th>Country </th><th>Area</th><th>Population</th><th>Density
    </th></tr>";
9  fseek($file, 35);
10 while ($data = fread($file, 35)) {
11     $fields = explode("|", $data);
12     echo "<tr><td>". $fields[0]. "</td><td>". $fields[1]. "</td><td>".
13         $fields[2]. "</td><td>". $fields[3]. "</td></tr>";
14 }
15 echo "</table></body></html>";
16 fclose($file);
17
18 ?>
```

El archivo de datos para el ejemplo anterior podría ser el siguiente. Note que este es un archivo de registros de tamaño fijo. Además, tome en cuenta que es necesario omitir los encabezados del archivo.

Belice	22966	334000	14.54
Costa_Rica	51100	4726000	92.49
El_Salvador	21041	6108000	290.29
Guatemala	108894	15284000	140.36
Honduras	112492	8447000	75.09
Nicaragua	129494	6028000	46.55
Panama	78200	3652000	46.70

7.12.3 Archivos de texto

Otra forma de leer archivos de texto es utilizar la función `file`, la cual transfiere un archivo completo a un arreglo. No es necesario abrir el archivo (`fopen`) para utilizar esta función.

```
1  <?php
2
3  $path = "/home/usr/data2.txt";
```

```

4  if (!file_exists($path))
5      exit("File not found");
6  $rows = file($path);
7  array_shift($rows);
8  echo "<html><body><table border=1>";
9  echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density
    </th></tr>";
10 foreach ($rows as $row) {
11     $fields = explode("|",$row);
12     echo "<tr><td>".$fields[0]."</td><td>".$fields[1]."</td><td>".
13         $fields[2]."</td><td>".$fields[3]."</td></tr>";
14 }
15 echo "</table></body></html>";
16
17 ?>

```

7.13 Archivos CSV

La función `fgetcsv` obtiene una línea del puntero a un archivo y la examina para tratar campos CSV. La función `fputcsv` da formato a una línea como CSV y la escribe en un puntero a un archivo.

```

1  <?php
2
3  $path = "/home/usr/data3.csv";
4  if (!file_exists($path))
5      exit("File not found");
6  $file = fopen($path, "r");
7  echo "<html><body><table border=1>";
8  echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density
    </th></tr>";
9  while ($fields = fgetcsv($file,"")) {
10     echo "<tr><td>".$fields[0]."</td><td>".$fields[1]."</td><td>".
11         $fields[2]."</td><td>".$fields[3]."</td></tr>";
12 }
13 echo "</table></body></html>";
14 fclose($file);
15
16 ?>

```

El archivo de datos para el ejemplo anterior podría ser el siguiente.

Belice	22966	334000	14.54
Costa_Rica	51100	4726000	92.49
El_Salvador	21041	6108000	290.29
Guatemala	108894	15284000	140.36
Honduras	112492	8447000	75.09
Nicaragua	129494	6028000	46.55
Panama	78200	3652000	46.70

7.13.1 Ejemplo

A continuación, un ejemplo de una aplicación Web en PHP para una lista de contactos. Cada contacto tiene un nombre, teléfono del trabajo, teléfono móvil, dirección electrónica y dirección postal. Se pueden agregar nuevos contactos, modificar los existentes, o eliminarlos.

Note que se utiliza borrado lógico en esta solución, es decir, se marcan los registros borrados y no se eliminan realmente del archivo. También, utilizamos un archivo con registros de tamaño fijo delimitados por el carácter "|".

```
1  <?php
2  $path = "data.txt";
3  if (file_exists($path))
4      $file = fopen($path, "r+");
5  else
6      $file = fopen($path, "a+");
7  while ($data = fread($file,154)) {
8      $array[] = explode('|',$data);
9  };
10
11 if (isset($_GET['get'])) {
12     $curr = (int)$_GET['get'];
13     $item = $array[$curr];
14 } else if (isset($_GET['delete'])) {
15     $curr = (int)$_GET['delete'];
16     fseek($file,$curr*154,SEEK_SET);
17     fwrite($file,'**deleted**');
18     $array[$curr][0] = '**deleted**';
19     $item = array(' ',' ',' ',' ',' ');
20     $curr = 0;
21 } else if (isset($_GET['save'])) {
22     $curr = (int)$_GET['save'];
23     $item = array(str_pad($_GET['name'],30),
24                  str_pad($_GET['work'],30),
25                  str_pad($_GET['mobile'],30),
26                  str_pad($_GET['email'],30),
27                  str_pad($_GET['address'],30));
28     fseek($file,$curr*154,SEEK_SET);
29     fwrite($file,implode('|',$item));
30     $array[$curr] = $item;
31 } else if (isset($_GET['append'])) {
32     $item = array(' ',' ',' ',' ',' ');
33     $curr = sizeof($array);
34 };
35
36 fclose($file);
37 ?>
38 <html>
39     <meta http-equiv="Content-Type" content="text/html; charset=UTF
40         -8" />
41     <head>
42         <title>Contacts</title>
43     </head>
44     <body>
```

```

44 <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
45 <div style="width: 250px; float: left;">
46 <h3>All Contacts</h3>
47 <table width="150" border=1>
48 <?php
49     for ($i=0;$i<sizeof($array);$i++) {
50         if (trim($array[$i][0])!="**deleted**")
51             echo '<tr><td><a href="?get='.$i.
52                 '" style="text-decoration:none;">'.
53                 $array[$i][0]. '</a></td></tr>';
54     }
55 <?>
56 </table>
57 </p>
58 <button name="append" value="">+</button>
59 </div>
60 <div style="margin-left:250px;">
61 <table>
62 <tr><td>&nbsp;</td></tr>
63 <tr><td><label>name</label></td><td>
64 <input name="name" size="30" value="<?php echo $item
65     [0]; ?>"/></td></tr>
66 <tr><td><label>work</label></td><td>
67 <input name="work" size="30" value="<?php echo $item
68     [1]; ?>"/></td></tr>
69 <tr><td><label>mobile</label></td><td>
70 <input name="mobile" size="30" value="<?php echo
71     $item[2]; ?>"/></td></tr>
72 <tr><td><label>email</label></td><td>
73 <input name="email" size="30" value="<?php echo $item
74     [3]; ?>"/></td></tr>
75 <tr><td><label>address</label></td><td>
76 <input name="address" size="30" value="<?php echo
77     $item[4]; ?>"/></td></tr>
78 <tr><td>&nbsp;</td></tr>
79 <tr><td><button name="delete" value="<?php echo $curr;
80     ?>">-</button></td>
81 <td align="right">
    <button name="save" value="<?php echo $curr; ?>">Save
    </button></td></tr>
</table>
</div>
</form>
</body>
</html>

```

7.13.2 Ejercicio propuesto

1. Modifique el ejemplo anterior para que utilice dos archivos, uno que almacene el nombre del contacto y un índice. Este índice indicará la posición en otro archivo en donde se encontrará el detalle del contacto.

2. Modifique el ejemplo para resolver el mismo problema pero utilizando registros de tamaño variable. Trate de solucionar de una forma “elegante” el borrado de registros.

Debido a que cada vez que se ejecuta la aplicación es necesario cargar todo el archivo en memoria, una solución es “paginar” los registros, es decir, cargar sólo una pequeña parte en memoria y permitir que el usuario cargue más registros conforme los necesite (posiblemente mediante un par de botones adicionales).

7.14 Manipulando XML y JSON

7.14.1 Lectura de datos XML

El método `XMLReader::open`, utilizado como método estático, establece el URL de entrada para el contenido XML que se procesará y `XMLReader::close` cierra dicha entrada. La función `XMLReader::read` mueve al siguiente nodo en el documento. Por su parte la función `XMLReader::next` mueve el cursor al siguiente nodo saltándose todos los subárboles.

Para obtener el valor de un atributo se utiliza la función `XMLReader::getAttribute(name)`. Para obtener el valor (texto) de un nodo se puede utilizar el atributo `XMLReader::value` y para conocer el nombre del elemento se utiliza el atributo `XMLReader::name`. El tipo del nodo se obtiene mediante `XMLReader::nodeType`. Algunos tipos de nodo son: `XMLReader::ELEMENT`, `XMLReader::TEXT`, `XMLReader::END_ELEMENT`.

Considere el siguiente archivo en formato XML con información sobre varios países:

```
1 <countries>
2   <country name="Belice" area="22966"
3     people="334000" density="14.54"/>
4   <country name="Costa Rica" area="51100"
5     people="4726000" density="92.49"/>
6   <country name="El Salvador" area="21041"
7     people="6108000" density="290.29"/>
8   <country name="Guatemala" area="108894"
9     people="15284000" density="140.36"/>
10  <country name="Honduras" area="112492"
11    people="8447000" density="75.09"/>
12  <country name="Nicaragua" area="129494"
13    people="6028000" density="46.55"/>
14  <country name="Panama" area="78200"
15    people="3652000" density="46.70"/>
16 </countries>
```

El siguiente programa genera una tabla en formato HTML a partir del anterior archivo de datos:

```
1 <?php
2
3 $path = "/Users/xxx/data.xml";
4
5 if (!file_exists($path))
6     exit("File not found");
7 $xml = XMLReader::open($path);
8 echo "<html><body><table border=1>";
9 echo "<tr><th>Country </th><th>Area </th><th>Population </th><th>Density
    </th></tr>";
```

```

10 while ($xml->read())
11 if ($xml->nodeType == XMLReader::ELEMENT && $xml->name == 'country')
12     {
13         $fields = array();
14         $fields[0] = $xml->getAttribute('name');
15         $fields[1] = $xml->getAttribute('area');
16         $fields[2] = $xml->getAttribute('people');
17         $fields[3] = $xml->getAttribute('density');
18         echo "<tr><td>".$fields[0]."</td><td>".$fields[1]."</td><td>".
19             $fields[2]."</td><td>".$fields[3]."</td></tr>";
20     }
21 echo "</table></body></html>";
22 $xml->close();
23 ?>

```

7.14.2 Escritura de datos mediante XML

La clase XMLWriter crea un objeto de escritura XML, y el método XMLWriter::openURI(path) establece el URI de salida para el contenido XML que se generará y XMLReader::flush escribe dicha entrada. La función XMLReader::startDocument crea el nodo principal de un documento xml, y el método XMLReader::endDocument finaliza el documento. Por su parte la función XMLReader::startElement crea un elemento XML y el método XMLReader::endElement lo finaliza.

Para escribir el valor de un atributo se utiliza la función XMLReader::writeAttribute(name,value). Para obtener el valor (texto) de un nodo se puede utilizar el atributo XMLReader::value.

El siguiente programa escribe una serie de datos al archivo XML de países:

```

1  <?php
2
3  $path = "/Users/xxx/data2.xml";
4
5  $writer = new XMLWriter();
6  $writer->openURI($path);
7  $writer->startDocument('1.0');
8
9  $writer->startElement('countries');
10
11 $writer->startElement('country');
12 $writer->writeAttribute('name', 'Belice');
13 $writer->writeAttribute('area', '22966');
14 $writer->writeAttribute('people', '334000');
15 $writer->writeAttribute('density', '14.54');
16 $writer->endElement();
17
18 $writer->startElement('country');
19 $writer->writeAttribute('name', 'Costa Rica');
20 $writer->writeAttribute('area', '51100');
21 $writer->writeAttribute('people', '4726000');
22 $writer->writeAttribute('density', '92.49');
23 $writer->endElement();
24
25 $writer->endElement();

```

```

26
27 $writer->endDocument();
28
29 $writer->flush();
30
31 ?>

```

7.14.3 Lectura de datos JSON

El método utilizado por PHP para tratar datos JSON es simplemente convertir Cadenas de texto (string) en formato JSON a arreglos de PHP. Para ello se utiliza la función `json_decode(string)` la cual recibe dicha cadena de texto y retorna el arreglo.

Por ejemplo considere el siguiente archivo de datos en formato JSON:

```

1  {"countries": [{"name":"Belice","area":"22966","people":"334000","
   density":"14.54"},
2    {"name":"Costa Rica","area":"51100","people":"4726000","density
   ":"92.49"},
3    {"name":"El Salvador","area":"21041","people":"6108000","density
   ":"290.29"},
4    {"name":"Guatemala","area":"108894","people":"15284000","density
   ":"140.36"},
5    {"name":"Honduras","area":"112492","people":"8447000","density
   ":"75.09"},
6    {"name":"Nicaragua","area":"129494","people":"6028000","density
   ":"46.55"},
7    {"name":"Panama","area":"78200","people":"3652000","density
   ":"46.70"}
8  ]}

```

Un programa para leer el anterior archivo y generar una tabla HTML con dicha información, sería el siguiente:

```

1  <?php
2
3  $path = "/Users/xxx/data.json";
4
5  if (!file_exists($path))
6      exit("File not found");
7
8  $data = file_get_contents($path);
9  $json = json_decode($data, true);
10
11 echo "<html><body><table border=1>";
12 echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density
   </th></tr>";
13 foreach ($json['countries'] as $row) {
14     echo "<tr><td>".$row['name']. "</td><td>".$row['area']. "</td><td>
   >".
15         $row['people']. "</td><td>".$row['density']. "</td></tr>";
16 }
17 echo "</table></body></html>";
18

```

7.14.4 Escritura de datos JSON

De igual forma para escribir datos en formato JSON, PHP utiliza la función `json_encode(arras)` la cual recibe un arreglo de PHP y retorna una cadena de texto en formato JSON.

El siguiente programa genera parte del archivo de datos de países en formato JSON:

```

1  <?php
2
3  $path = "/Users/xxx/data2.json";
4  $file = fopen($path, "w");
5
6  $countries = array(
7      array("name"=>"Belice", "area"=>"22966", "people"=>"334000", "
          density"=>"14.54"),
8      array("name"=>"Costa Rica", "area"=>"51100", "people"=>"4726000", "
          density"=>"92.49")
9  );
10
11 $json = json_encode($countries);
12 fwrite($file, $json);
13 fclose($file);
14
15 ?>
```

7.15 Generación de imágenes

PHP permite la generación de imágenes mediante una serie de primitivas de dibujo. El primer paso para realizar esto consiste en definir el tamaño de la imagen mediante la instrucción `imagecreate(width,height)`. Luego se aplican sobre dicha imagen las primitivas de dibujo deseadas, al final se debe usar una instrucción para convertir la imagen en algún formato conocido como: `imagepng(image)`, `imagegif(image)` ó `imagejpeg(image)`. Posteriormente, se puede destruir la imagen en memoria utilizando la instrucción `imagedestroy(image)`. Si la imagen se desea desplegar en el navegador es necesario enviar el encabezado adecuado, por ejemplo:

```
header( "Content-type: image/png" )
```

7.15.1 Definición de colores

Para definir los colores que se utilizarán con las diferentes primitivas de dibujo es necesario utilizar la instrucción `imagecolorallocate(image, red, green, blue)`. Los rangos de los valores de rojo, verde y azul deben estar entre 0 y 255. Estos valores también pueden ser especificados en hexadecimal, por ejemplo: `0xFF`

Es importante indicar que la primer llamada que se realice a esta función definirá automáticamente el color de fondo (background) de la imagen, por ejemplo:

```

1  <?php
2      $myImage = imagecreate( 200, 100 );
3      $myGray = imagecolorallocate( $myImage, 204, 204, 204 );
```



```

4     header( "Content-type: image/png" );
5     imagepng( $myImage );
6     imagedestroy( $myImage );
7  ?>

```

7.15.2 Primitivas de dibujo

Se cuenta con diferentes primitivas de dibujo que permiten crear gráficos sobre la imagen. Existen primitivas para dibujar el borde de diferentes geometrías:

Figura Instrucción Línea `imageline(image, posX1, posY1, posX2, posY2, color)` Rectángulo `imagerectangle(image, minX, minY, maxX, maxY, color)` Elipse `imageellipse(image, posX, posY, width, height, color)` Polígono `imagepolygon(image, array, num_points, color)` Para dibujar geometrías rellenas se deben usar las sentencias:

Figura Instrucción Rectángulo `imagefilledrectangle(image, minX, minY, maxX, maxY, color)` Elipse `imagefilledellipse(image, posX, posY, width, height, color)` Polígono `imagefilledpolygon(image, array, num_points, color)`

```

1  <?php
2      $myImage = imagecreate( 200, 100 );
3      $myGray = imagecolorallocate( $myImage, 204, 204, 204 );
4      $myBlack = imagecolorallocate( $myImage, 0, 0, 0 );
5      imageline( $myImage, 15, 35, 120, 60, $myBlack );
6      imagerectangle( $myImage, 15, 35, 120, 60, $myBlack );
7      imageellipse( $myImage, 90, 60, 160, 50, $myBlack );
8      $myPoints = array( 20, 20, 185, 55, 70, 80 );
9      imagepolygon( $myImage, $myPoints, 3, $myBlack );
10     header( "Content-type: image/png" );
11     imagepng( $myImage );
12     imagedestroy( $myImage );
13  ?>

```

7.3. Manejo de texto Es posible incluir texto en las imágenes para lo cual se puede utilizar la instrucción `imagestring(image, font, posX, posY, string, color)`. El tipo de fuente (font) puede ser un número entre 1 y 5 para los tipos de fuente predefinidos. Es posible cargar nuevas fuentes de letra desde archivos en formato gdf mediante la instrucción `imageloadfont(filename)`. También se puede obtener el ancho y tamaño del tipo de letra actual mediante las sentencias `imagefontwidth()` e `imagefontheight()`.

```

1  <?php
2      $textImage = imagecreate( 200, 100 );
3      $white = imagecolorallocate( $textImage, 255, 255, 255 );
4      $black = imagecolorallocate( $textImage, 0, 0, 0 );
5      $yOffset = 0;
6      for ( $i = 1; $i <= 5; $i++ ) {
7          imagestring( $textImage, $i, 5, $yOffset, "This is system font $i",
8              $black
9          );
10         $yOffset += imagefontheight( $i );
11     }
12     header("Content-type: image/png");
13     imagepng( $textImage );
14     imagedestroy( $textImage );

```

```

14  ?>
15  En ocasiones es conveniente utilizar tipos de letra TrueType que
    mejoren la apariencia de la imagen. Para cargar archivos de este
    tipo se utiliza la instrucción imagefttext(image, size, angle,
    posX, posY, color, filename, string)
16  \begin{lstlisting}
17  <?php
18      $textImage = imagecreate( 200, 120 );
19      $white = imagecolorallocate( $textImage, 255, 255, 255 );
20      $black = imagecolorallocate( $textImage, 0, 0, 0 );
21      imagefttext( $textImage, 16, 0, 10, 50, $black, "fonts/truetype
    /
22      ttf-bitstream-vera/Vera.ttf", "Vera, 16 pixels" );
23      header( "Content-type: image/png" );
24      imagepng( $textImage );
25      imagedestroy( $textImage );
26  ?>

```

7.16 Plantillas

Mustache es un motor de plantillas para PHP, es decir, separa el código PHP, como lógica de negocios, del código HTML, como lógica de presentación, y genera contenidos web mediante la colocación de etiquetas mustache en un documento. Mustache está disponible en muchos lenguajes de programación y es consistente entre plataformas.

La versión para PHP puede ser descargada desde <https://github.com/bobthecow/mustache.php>. Únicamente es necesario copiar, en el directorio en donde se correrá el programa, el directorio src/mustache.

7.16.1 Uso básico

Un programa PHP debe realizar la inicialización del motor mustache mediante el autoloader que viene incluido con el código. En dicha inicialización se puede indicar la extensión y ubicación de los archivos de plantillas, por ejemplo /views. Dentro del programa la plantilla se carga mediante el método loadTemplate y luego se puede aplicar mediante la función render. Note que a esta última función se deben pasar como parámetros, mediante un arreglo, el conjunto de datos que serán utilizados por la plantilla.

```

1  <?php
2
3  require 'Mustache/Autoloader.php';
4  Mustache_Autoloader::register();
5
6  $mustache = new Mustache_Engine(array(
7      'loader' => new Mustache_Loader_FilesystemLoader(
8          dirname(__FILE__) . '/views',
9          array('extension' => '.tpl')),
10 ));
11
12 $path = "data.json";
13 if (!file_exists($path))

```

```

14     exit("File not found");
15
16     $data = file_get_contents($path);
17     $json = json_decode($data, true);
18
19     $tpl = $mustache->loadTemplate('Example9_1.tpl');
20     echo $tpl->render(array('countries' => $json['countries']));
21
22     ?>

```

El mecanismo de marcado (etiquetas) de mustache utiliza pares de corchetes para definir el inicio y final de cada bloque de elementos. Por ejemplo, para el programa anterior la plantilla se vería de la siguiente forma. Recuerde que esta plantilla tiene como nombre Example9_1.tpl y se encuentra bajo el directorio /views

```

1 <html><body><table border=1>
2 <tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th>
   </tr>
3 {{#countries}}
4     <tr><td>{{name}}</td><td>{{area}}</td><td>
5         {{people}}</td><td>{{density}}</td></tr>
6 {{/countries}}
7 </table></body></html>

```

Es importante indicar que el lenguaje de marcado de mustache es consistente entre las distintas implementaciones de la librería, en diferentes lenguajes de programación. Por lo tanto no se puede incluir código PHP en una plantilla de mustache, pero dicha plantilla puede ser "transportada" a otro ambiente de programación como: javascript, python, Java, C++ ó ASP; y seguirá funcionando de la misma forma.

7.16.2 Condicionales

Es posible indicar que cierto contenido debe aparecer en la salida bajo ciertas condiciones y otro no. Para eso se utiliza combinaciones de etiquetas con "#z". Por ejemplo considere el siguiente programa que utiliza algunos datos de países, pero algunos están incompletos.

```

1 <?php
2
3 require 'Mustache/Autoloader.php';
4 Mustache_Autoloader::register();
5
6 $mustache = new Mustache_Engine(array(
7     'loader' => new Mustache_Loader_FilesystemLoader(
8         dirname(__FILE__) . '/views',
9         array('extension' => '.tpl')),
10 ));
11
12 $countries = array(
13     array("name"=>"Belice", "area"=>"22966", "people"=>"334000", "
14         density"=>"14.54"),
15     array("area"=>"33444", "people"=>"3434340", "density"=>"0"),
16     array("name"=>"Costa Rica", "area"=>"51100", "people"=>"4726000", "
17         density"=>"92.49"),

```

```

16     array("area"=>"229656","people"=>"99800","density"=>"0"),
17 );
18
19 $tpl = $mustache->loadTemplate('Example9_2.tpl');
20 echo $tpl->render(array('countries' => $countries));
21
22 ?>

```

La plantilla asociada, llamada Example9_2.tpl, mostraría una etiqueta "desconocido.^{en} aquellos registros que no aparezca su nombre, tal como se muestra a continuación:

```

1 <html><body><table border=1>
2 <tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th>
3 </tr>
4 {{#countries}}
5   {{#name}}
6     <tr><td>{{name}}</td><td>{{area}}</td><td>
7       {{people}}</td><td>{{density}}</td></tr>
8     {{/name}}
9     {{^name}}
10      <tr><td>Desconocido</td><td>{{area}}</td><td>
11        {{people}}</td><td>{{density}}</td></tr>
12      {{/name}}
13    {{/countries}}
14  </table></body></html>

```

7.17 Servicios Web tipo RESTfull

La librería ToroPHP permite la creación de servicios web tipo RESTfull de forma sencilla y eficiente en PHP. Esta se encuentra disponible en <http://github.com/anandkunal/ToroPHP>. Una vez que se descarga el archivo Toro.php éste puede ser probado utilizando un pequeño programa de ejemplo, llamado prueba.php:

```

1 <?php
2
3 require("Toro.php");
4
5 class HelloHandler {
6     function get() {
7         echo "Hello, world";
8     }
9 }
10
11 Toro::serve(array(
12     "/" => "HelloHandler",
13 ));
14
15 ?>

```

Dicho programa se puede ejecutar mediante el url <http://localhost/websevice/prueba.php/>

7.17.1 Consulta de datos

En el protocolo RESTfull se utiliza el método GET para recuperar información. Para identificar el tipo de elemento a consultar se puede utilizar un "subdirectorio.^{en} el URL.

Por ejemplo, considere el siguiente programa que recupera todos datos de los países desde la base de datos y los presenta en formato JSON. Para invocarlo se debe usar un URL como `http://localhost/prueba.php/country`.

```
1 <?php
2 require("Toro.php");
3
4 class DBHandler {
5     function get() {
6         try {
7             $dbh = new PDO('sqlite:test.db');
8         } catch (Exception $e) {
9             die("Unable to connect: " . $e->getMessage());
10        }
11        try {
12            $stmt = $dbh->prepare("SELECT * FROM countries");
13            $stmt->execute();
14
15            $data = Array();
16            while ($result = $stmt->fetch(PDO::FETCH_ASSOC)) {
17                $data[] = $result;
18            }
19            echo json_encode($data);
20        } catch (Exception $e) {
21            echo "Failed: " . $e->getMessage();
22        }
23    }
24 }
25
26 Toro::serve(array(
27     "/country" => "DBHandler",
28 ));
29 ?>
```

Si se desea invocar a este servicio web desde otro programa PHP basta con definir el contenido a partir de la dirección URL del servicio, por ejemplo:

```
1 <?php
2
3     $path = "http://localhost/prueba.php/country";
4
5     $data = file_get_contents($path);
6     $json = json_decode($data, true);
7
8     echo "<html><body><table border=1>";
9     echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>
    Density</th></tr>";
10    foreach ($json as $row) {
11        echo "<tr><td>".$row['name']. "</td><td>".$row['area']. "</td><
    td>".
```

```

12         $row['population']. "</td><td>". $row['density']. "</td></tr>";
13     }
14     echo "</table></body></html>";
15
16 ?>

```

7.17.2 Paso de parámetros

En los servicios web tipo RESTfull, y en ToroPHP en particular, los parámetros de la consulta son pasados también como se fueran subdirectorios y no mediante el símbolo ?. Por ejemplo, una llamada que normalmente se realizaría de la siguiente forma:

```
1 | http://localhost/prueba.php?table=country\&name=Nicaragua
```

se realiza de la siguiente forma utilizando un servicio RESTfull:

```
1 | http://localhost/prueba.php/country/Nicaragua
```

note que en este caso el nombre de los parámetros no aparecen y ellos deben ser identificados de forma implícita por su posición en la solicitud.

Por ejemplo, considere ahora una modificación al programa anterior que recuperará los datos de un país en la base de datos y los presenta en formato JSON. Para invocarlo, utilice un URL como <http://localhost/prueba.php/country/Nicaragua>.

```

1 <?php
2     require("Toro.php");
3
4     class DBHandler {
5
6         function get($name=null) {
7             try {
8                 $dbh = new PDO('sqlite:test.db');
9             } catch (Exception $e) {
10                 die("Unable to connect: " . $e->getMessage());
11             }
12             try {
13                 if ($name!=null) {
14                     $stmt = $dbh->prepare("SELECT * FROM countries
15                                     WHERE name = :name");
16                     $stmt->bindParam(':name', $name, PDO::PARAM_STR);
17                 } else {
18                     $stmt = $dbh->prepare("SELECT * FROM countries");
19                 }
20                 $stmt->execute();
21
22                 $data = Array();
23                 while ($result = $stmt->fetch(PDO::FETCH_ASSOC)) {
24                     $data[] = $result;
25                 }
26                 echo json_encode($data);
27             } catch (Exception $e) {
28                 echo "Failed: " . $e->getMessage();
29             }
30         }
31     }
32 }

```

```

29     }
30 }
31
32 Toro::serve(array(
33     "/country" => "DBHandler",
34     "/country/:alpha" => "DBHandler",
35 ));
36 ?>

```

Note que existen tres tipos de parámetros que reconoce ToroPHP: number, alpha y string; o bien se puede utilizar una expresión regular como: `([0-9]+)`, `([a-zA-Z0-9-_\]+)` o `([a-zA-Z]+)`. Note que pueden ser utilizados múltiples parámetros en la solicitud, y estos serán pasados en el orden en que aparecen al método get de la clase utilizada como manejador (handler).

7.17.3 Envío de datos

Para enviar información a un servicio RESTfull se utiliza el método POST o PUT. Generalmente el método PUT se utiliza para crear un elemento, y el método POST para modificar los datos de un elemento existente. Una nueva modificación al servicio web incorpora la capacidad de modificar los datos de un registro, tal como se muestra a continuación:

```

1  <?php
2      require("Toro.php");
3
4      class DBHandler {
5
6          function get($name=null) {
7              // como en el ejemplo anterior
8          }
9
10         function post($name=null) {
11             try {
12                 $dbh = new PDO('sqlite:test.db');
13             } catch (Exception $e) {
14                 die("Unable to connect: " . $e->getMessage());
15             }
16             try {
17                 $area = $_POST['area'];
18                 $population = $_POST['population'];
19                 $density = $_POST['density'];
20                 echo $area;
21
22                 $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::
23                     ERRMODE_EXCEPTION);
24
25                 $stmt = $dbh->prepare("UPDATE countries SET area=:area,
26                                     population=:population, density=:
27                                     density
28                                     WHERE name = :name");
29                 $stmt->bindParam(':area', $area);
30                 $stmt->bindParam(':population', $population);
31                 $stmt->bindParam(':density', $density);

```

```

31         $dbh->beginTransaction();
32         $stmt->execute();
33         $dbh->commit();
34         echo 'Successfull';
35     } catch (Exception $e) {
36         $dbh->rollBack();
37         echo "Failed: " . $e->getMessage();
38     }
39 }
40 }
41
42 Toro::serve(array(
43     "/country" => "DBHandler",
44     "/country/:alpha" => "DBHandler",
45 ));
46 ?>

```

Sin embargo para invocar esta función del servicio web, desde otro programa PHP, es necesario utilizar un stream PHP tal como se muestra a continuación:

```

1  <?php
2
3  $name='Nicaragua';
4  $url = 'http://localhost/prueba.php/country/'. $name;
5
6  $data = array('name'=>'Nicaragua','area'=>'129000',
7               'population'=>'6548000','density' => '46.55');
8  $options = array(
9      'http' => array(
10         'header' => "Content-type: application/x-www-form-urlencoded
11         \r\n",
12         'method' => 'POST',
13         'content' => http_build_query($data),
14     )
15 );
16 $context = stream_context_create($options);
17 $result = file_get_contents($url, false, $context);
18
19 echo $result;
20 ?>

```


Capítulo 8

Conexión con bases de datos

Hasta la versión 5.5 de PHP se usaban tres extensiones para mysql. La extensión MySQL, MySQLi y PDO. Actualmente, la primera de ellas ha quedado obsoleta y no se usa más (PHP >=5.5). Si la usamos se genera error de tipo E_DEPRECATED.

Las principales ventajas que proporciona MySQLi respecto al uso de la extensión MySQL son:

1. Interfaz orientada a objetos. También podemos usar la interfaz procedimental.
2. Soporte para declaraciones preparadas
3. Posibilidad de usar transacciones.

8.1 MySQLi

La extensión MySQLi proporciona procedimientos y propiedades de objeto que informan del último error producido en la base de datos. El uso de cada uno depende de si está usando la interfaz orientada a objetos o la interfaz procedimental.

Veamos la tabla siguiente: Para realizar la conexión almacena la información necesaria en constantes que faciliten su posterior modificación.

```
1 | define("DB_HOST","localhost" );  
2 | define("DB_USER", "usuario");  
3 | define("DB_PASS", "mypassword");  
4 | define("DB_DATABASE", "Demo" );
```

8.1.1 Interfaz orientada a objetos

Para la conexión crea una instancia de mysqli pasando los datos necesarios para la conexión: host, usuario, contraseña y nombre de la base de datos.

```
1 | $mysqli = new mysqli(DB_HOST, DB_USER, DB_PASS, DB_DATABASE);  
2 | if($mysqli->connect_errno > 0){  
3 | die("Imposible conectarse con la base de datos [" . $mysqli->  
   |   connect_error . "]);  
4 | }
```

Orientado a objetos (propiedades)	Procedimientos	Descripción
<code>string \$mysqli->error;</code> donde \$mysqli es una instancia de conexión con el servidor mysql.	<code>string mysqli_error(mysqli \$link);</code> mysqli \$link es una instancia de conexión con el servidor mysql.	Devuelve el último mensaje de error generado en la base de datos.
<code>int \$mysqli->errno;</code>	<code>int mysqli_errno(mysqli \$link);</code>	Devuelve el código de error generado en la base de datos
<code>string \$mysqli->connect_error;</code>	<code>string mysqli_connect_error();</code>	Devuelve el mensaje de error generado por el último intento fallido de conexión.
<code>int \$mysqli->connect_errno;</code>	<code>int mysqli_connect_error();</code>	Devuelve el código de error generado por el último intento fallido de conexión.

También es posible no indicar la base de datos al crear la instancia y hacerlo posteriormente mediante `$mysqli -> select_db()`. Este método devuelve un boolean para verificar el posible error al seleccionar la base de datos.

```
1 | $mysqli = new mysqli(DB_HOST, DB_USER, DB_PASS);
2 | if (!$mysqli->select_db(DB_DATABASE)) die ($mysqli->error);
```

8.1.2 Interfaz procedimental

Creemos la conexión mediante el procedimiento `mysqli_connect()` que tiene como parámetros los datos correspondientes y devuelve un objeto que representa la conexión al servidor mysql.

```
1 | $con = mysqli_connect(DB_HOST, DB_USER, DB_PASS, DB_DATABASE);
2 | if (mysqli_connect_errno()) {
3 |     echo "Imposible conectarse a la base de datos: " .
        mysqli_connect_error();
4 | } else {
5 | }
```

Errores

Antes de pasar a interactuar con MySQL, hay que comentar ciertas funciones que proporcionan información de los errores que se pueden producir en las diferentes etapas de dicha interacción.

La extensión MySQLi proporciona procedimientos y propiedades de objeto que informa del último error producido en la base de datos. El uso de cada uno depende de si está usando la interfaz orientada a objetos o la procedimental.

Vamos a ver los principales:

Orientado a objetos (propiedades)	Procedimientos	Descripción
string \$mysqli->error; donde \$mysqli es una instancia de conexion con el servidor mysql.	donde string mysqli_error(mysqli \$link); mysqli \$link es una instancia de conexion con el servidor mysql.	Devuelve el último mensaje de error generado en la base de datos.
int \$mysqli->errno;	int mysqli_errno(mysqli \$link);	Devuelve el código de error generado en la base de datos
string \$mysqli->connect_error;	string mysqli_connect_error();	Devuelve el mensaje de error generado por el último intento fallido de conexión.
int \$mysqli->connect_errno;	int mysqli_connect_error();	Devuelve el código de error generado por el último intento fallido de conexión.

8.2 Métodos GET, POST y ENLACE

Los formularios html aumentan la interactividad de nuestra Web y reciben información de los usuarios del sitio. Los formularios html están compuestos por campos de texto y botones. Cuando un usuario llena el formulario de una aplicación web, presiona "enviar"(botón) para enviar los datos al servidor de alguna manera. Consideremos las dos formas de envío de datos: método POST o GET.

Por ejemplo: `<form action="nuevousuario.php" method="get">` es el encabezado de un formulario con el método get.

En la linea anterior, cuando el usuario presiona el botón "Enviar", el atributo `action` indica el tipo de acción que realiza el formulario, invoca la url del programa `nuevousuario.php`. El valor `get` es el valor por defecto. Si no concretamos el `method`, la información se envia a través de este medio. Mediante el atributo `method` le indicamos al formulario la forma de envio de los datos. `get` indica que los datos enviados se adjuntan en la barra de direcciones del cliente, al final de la url correspondiente y después de un signo de interrogación de cierre. Si se envía más de un dato, éstos van separados por el símbolo `&`.

Veamos el aspecto de un formulario cualquiera para hacernos una idea general.

Este formulario consta de varios campos que se solicitan al usuario como Nombre, Apellidos, Correo electrónico, País y Mensaje. Posiblemente los nombres de los campos en el código HTML sean del tipo `nombre_user`, `apellidos_user`, `email_user`, `pais_user` y `msg`.

Un formulario se escribe en HTML. Este es el código de un formulario:

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Ejemplo del uso de formularios - jodocha.com</title>
5 </head>
```

```

6 <body>
7 <form method="get" action="action.php">
8 Selecciona tus intereses:
9 <br />
10 <input name="cbipeliculas" type="checkbox" />Películas
11 <br />
12 <input name="cbilibros" type="checkbox" checked="checked" />Libros
13 <br />
14 <input name="cbiinternet" type="checkbox" />Internet
15 </form>
16 </body>
17 </html>

```

La diferencia entre los métodos get y post radica en la forma de enviar los datos a la página cuando presiona el botón "Enviar". El método GET envía los datos usando la URL o públicos, el POST los envía de forma privada.

Un resultado usando el método GET es el siguiente: <http://jodocha.com/newuser.php?nombre=Pepe&apellido=Flores&email=h52turam%40uco.es&sexo=Mujer>

En esta URL distinguimos varias partes: <http://jodocha.com/newuser.php> es la dirección web en sí.

El símbolo ? indica dónde empiezan los parámetros que se reciben desde el formulario que ha enviado los datos a la página.

Las parejas dato1=valor1, dato2=valor2, dato3=valor3... reflejan el nombre y el valor de los campos enviados por el formulario.

Por ejemplo: nombre=Pepe, apellidos=Flores, etc. dice que el campo del formulario que se denomina nombre llega con valor "Pepe" mientras que el campo del formulario denominado apellidos llega con valor "Flores".

Ten en cuenta que para separar la primera pareja se usa el símbolo "?&" para las restantes el símbolo "&".

Otro aspecto a tener en cuenta es que determinados caracteres no son recibidos en la URL de la misma forma exactamente en que fueron escritos en el formulario. El valor del campo email que se recibe en la URL es `h52turam%40uco.es`, mientras que el usuario en el formulario escribió `h52turam@uco.es`. Como vemos, el carácter `@` ha sido sustituido por los caracteres `%40`. Estas equivalencias se introducen automáticamente en la transmisión de datos debido a que las URLs no admiten determinados caracteres como letras con tildes, arrobas y otros. No debes preocuparte por esta codificación, ya que si posteriormente rescatamos los valores mediante otros mecanismos volveremos a obtener el texto original. Simplemente, conviene conocer esta circunstancia para no pensar que están ocurriendo cosas extrañas o errores.

Hemos visto el resultado de un envío por el método get. En el caso de un envío de datos usando el método POST, aunque estos datos también son enviados (de forma "oculta"), no los vemos en la URL. Para recuperar los valores de los campos en el caso de un envío con el método post necesitamos otras herramientas (como valores del lenguaje PHP para recuperar el valor de esos campos).

El resultado final con ambos métodos es el mismo: la información se transmite de un lado a otro. La diferencia radica en que con el método GET vemos los parámetros pasados ya que están dentro de la URL mientras que con el método POST los parámetros quedan ocultos y para rescatarlos hay que usar otras herramientas.

Un ejemplo de uso del método post es este: `<form action="http://jodocha.com/prog/newuser.php" method="post">`

Cuando usamos el método post los datos enviados desde el formulario no son visibles, pero sí son recuperables usando las instrucciones adecuadas.

8.2.1 Ejercicio propuesto

1. Crea un archivo HTML para un formulario que contenga un checkbox (Desea recibir factura?), dos input radio (Hombre/Mujer), tres input tipo texto (Nombre, Dirección, Email) y un combobox para forma de envío (normal, por avión o urgente), junto con un botón de envío. Establece como url de envío la dirección recibe.php y método de envío get y comprueba que se recuperan los datos enviados correctamente a través de la url.

8.2.2 Variables \$_REQUEST, \$_GET y \$_POST

Cuando un usuario presiona el botón "enviar" de un formulario, la información que contienen sus campos es enviada a una dirección URL desde la que recuperamos para tratarla de alguna manera. Si realiza una compra, recuperamos los datos para completar el proceso de pago. La información del formulario "viaja" almacenada en variables que recuperamos y utilizamos mediante PHP.

La recuperación de variables con PHP es fácil, ya sean datos de un formulario enviado por el método post (parámetros no visibles) o por el método get (parámetros en la URL sí visibles). Aclaremos que hay varios métodos para recuperar variables con PHP. Para los formularios tenemos los métodos GET, POST, y REQUEST.

8.2.2.1 Recuperar variables con GET

GET recupera parámetros desde la URL o desde formularios enviados con el método GET. Escribe este código y guárdalo como ejemplo1.html. A continuación, ejecuta el archivo en el servidor, introduce un nombre en el campo del formulario y visualiza el resultado.

```
1 <form name="formulario" method="get" action="ejemploGet.php">
2 Nombre: <input type="text" name="nombre" value="">
3 <input type="submit"/>
4 </form>
```

Antes de presionar el botón "enviar", creamos el archivo ejemploGet.php que es el indicado en el formulario para enviar de los datos.

Escribe este código y guárdalo como ejemploGet.php. A continuación, sube el archivo al servidor en la misma carpeta donde subiste el ejemplo1.html

```
1 <?php //Ejemplo jodocha.com
2 $nombre = $_GET['nombre'];
3 echo$nombre;
4 ?>
```

La instrucción `$_GET['nombreDelParametro']` recupera la información recibida.

Una vez completado el paso anterior, presionamos el botón "enviar" del ejemplo1.html. Vamos a explicar el proceso.

El primer archivo es un documento HTML. VeamosPara ser más correctos, deberíamos haberlo escrito de esta manera:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Ejemplo jodocha.com</title>
5 </head>
6 <body>
7 <form name="formulario"method="get"action="ejemploGet.php">
8 Nombre: <input type="text"name="nombre"value="">
9 <input type="submit"/>
10 </form>
11 </body>
12 </html>
```

Recordar que los navegadores interpretan y muestran el resultado de un código HTML (o PHP) incluso cuando la sintaxis o la definición del documento no es correcta.

En el ejemplo definimos un formulario con el método de envío de datos GET y URL de destino ejemploGet.php. El formulario tiene un campo cuyo atributo name es "nombre"que define a la variable a recuperar en la URL de destino.

En el archivo ejemploGet.php incluimos `$nombre = $_GET['nombre'];` con la que creamos una variable php denominada `$nombre` donde almacenamos la información del campo 'nombre' que recibe la URL a través del método get proveniente del formulario. Para otros campos definidos como apellidos, teléfono, edad, el tratamiento es similar. Veamos:

```
1 $apellidos = $_GET['apellidos'];
2 $telefono = $_GET['celular'];
3 $edadPersona = $_GET['edad'];
```

Atento a que una cosa es la variable en que almacenamos la información recuperada, y otra es el nombre del campo del formulario de donde proviene. En `$apellidos = $_GET['apellidos'];` coinciden el nombre de la variable que utilizamos con el nombre de campo del formulario. Sin embargo, en `$telefono = $_GET['celular'];` no coinciden. En este caso, el campo que proviene del formulario se llama celular"mientras que la información proveniente de ese campo la almacenamos en una variable llamada `$telefono`. Finalmente, en `$edadPersona = $_GET['edad'];` almacenamos en la variable `$edadPersona` información proveniente de un campo del formulario denominado 'edad'.

Con frecuencia los nombres de las variables y de los campos del formulario coinciden, pero en otras ocasiones no. Esto queda a elección del programador.

Como observamos, recuperar datos enviados por un formulario con el método GET es simple usando PHP.

8.2.3 Ejercicio Resuelto

1. Diseñar un formulario web que solicite la altura y el diámetro de un cilindro en metros. Una vez que el usuario introduzca los datos y presione el botón calcular, el programa calcula el volumen del cilindro y muestra el resultado en el navegador.
2. Esquematizar la solución en pseudocódigo es una buena idea antes de realizar la programación, pues define conceptualmente el código antes de escribirlo. Es adecuado para personas

que se están iniciando en la programación.

En primer lugar, creamos el archivo html con el formulario.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Ejemplo jodocha.com</title>
5 <meta charset="utf-8">
6 </head>
7 <body>
8 <form name="formularioDatos"method="get"action="ejemploGet1.php">
9 <p>CÁLCULO DEL VOLUMEN DE UN CILINDRO </p>
10 <br/>
11 Introduzca el diámetro en metros: <input type="text"name="diam"value
    ="">
12 <br/> <br/>
13 Introduzca la altura en metros: <input type="text"name="altu"value
    ="">
14 <br/> <br/>
15 <input value="Calcular" type="submit" />
16 </form>
17 </body>
18 </html>
```

Por otro lado, creamos el archivo php con el tratamiento de datos:

```
1 <?php //Ejemplo
2 $diametro = $_GET['diam'];
3 $altura = $_GET['altu'];
4 $radio = $diametro/2;
5 $Pi = 3.141593;
6 $volumen = $Pi*$radio*$radio*$altura;
7 echo"<br/> &nbsp; El volumen del cilindro es de". $volumen. "metros c
    úbicos";
8 ?>
```

Hemos escrito la potencia del radio como `$radio * $radio`. En otros lenguajes existe el operador exponente, en php esta operación se realiza recurriendo a `$radio^2`.

Obtenemos un resultado. Haz pruebas introduciendo como valores de diámetro y altura 2,15 y 1,75 en vez de 2.15 y 1.75. Posiblemente no obtengas un resultado adecuado si usas las comas, ya que en PHP el separador de la parte decimal de un número es el punto.

8.2.4 Ejercicio Resuelto

1. Diseñar un desarrollo web con php que pida al usuario el precio de tres productos en tres establecimientos distintos denominados "Tienda 1", "Tienda 2" y "Tienda 3". Una vez introducida esta información debe calcular y mostrar el precio medio del producto.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Ejemplo jodocha.com</title>
```

```

5 <meta charset="utf-8">
6 </head>
7 <body>
8 <form name="formularioDatos" method="get" action="ejemploGet2.php">
9 <p> CÁLCULO DEL PRECIO MEDIO DE UN PRODUCTO </p>
10 <br/>
11 Introduzca el precio del producto en el establecimiento 1, en Bolí
   vares: <input type="text" name="precio1" value="">
12 <br/> <br/>
13 Introduzca el precio del producto en el establecimiento 2, en Bolí
   vares: <input type="text" name="precio2" value="">
14 <br/> <br/>
15 Introduzca el precio del producto en el establecimiento 3, en Bolí
   vares: <input type="text" name="precio3" value="">
16 <br/> <br/>
17 <input value="Calcular" type="submit" />
18 </form>
19 </body>
20 </html>

```

Es posible que durante el desarrollo del programa encuentres visualizaciones de este tipo, donde compruebas que los acentos o tildes no se ven bien.

Recordarte que la solución a esto es, cuando estamos trabajando con nano, elegir en el menú Formato la opción `Çodificar en UTF-8 sin BOM`". En caso de que por error el archivo esté en otro formato, elige la opción `Çonvertir en UTF-8 sin BOM`" para dejar correctamente la codificación del archivo. También es necesario introducir la etiqueta: `<meta charset="utf-8">` entre las etiquetas `<head>` `\dots` `</head>` para lograr que la visualización sea correcta.

Por otro lado necesitamos el archivo php para mostrar los resultados. El código es:

```

1 <?php //Ejemplo jodocha.com
2 $precio1 = $_GET['precio1'];
3 $precio2 = $_GET['precio2'];
4 $precio3 = $_GET['precio3'];
5 $media = ($precio1+$precio2+$precio3)/3;
6 echo "<br/> &nbsp; DATOS RECIBIDOS";
7 echo "<br/> &nbsp; Precio producto establecimiento 1: ". $precio1. "
   Bolívares";
8 echo "<br/> &nbsp; Precio producto establecimiento 2: ". $precio2. "
   Bolívares";
9 echo "<br/> &nbsp; Precio producto establecimiento 2: ". $precio3. "
   Bolívares <br/>";
10 echo "<br/> &nbsp; El precio medio del producto es de ". $media. "
   Bolívares";
11 ?>

```

8.2.5 ¿Desde donde se recuperan los datos?

Si has realizado el ejercicio anterior habrás comprobado que una vez se presiona el botón de envío del formulario la URL es de este tipo:

`http://www.jodocha.com/ejemploGet2.php?precio1=18.55&precio2=21&precio3=27.55`

Aquí comprobamos que los datos están en la URL. Ahora bien, los datos venían del formulario.

¿dónde se están recuperando los datos? Para responder a esta pregunta escribe una URL de este tipo:

`http://www.jodocha.com/ejemploGet2.php?precio1=10&precio2=15&precio3=20`

8.2.6 Datos recibidos

Vemos que:

Precio producto establecimiento 1: 10 Bolívares

Precio producto establecimiento 2: 15 Bolívares

Precio producto establecimiento 2: 20 Bolívares

El precio medio del producto es de 15 Bolívares

Conclusión: los datos pasan del formulario a la URL, y desde la URL son recuperados por el intérprete PHP usando `$_GET['nombreDelParametro']`. Por tanto los datos no vienen directamente del formulario, sino que son recuperados a través de la URL.

8.2.7 Ejercicio propuesto

1. Diseñar un desarrollo web simple con PHP que dé respuesta a la necesidad que se plantea a continuación:

Un operario de una fábrica recibe cada cierto tiempo un depósito cilíndrico de dimensiones variables que llena de aceite a través de una toma con cierto caudal disponible. Se desea crear una aplicación web que le indique cuánto tiempo transcurre hasta el llenado del depósito. El caudal disponible se considera estable para el tiempo de llenado del depósito y lo facilita el propio operario, aportando el dato en litros por minuto.

8.2.8 Redefinamos los conceptos

El concepto GET es obtener información del servidor. Traer datos que están almacenadas en el servidor, ya sea una base de datos o archivo al cliente. Independientemente de que para eso tengamos que enviar (request) algún dato que será procesado para luego devolver la respuesta (response) que esperarnos, un ejemplo sería recibir un identificador para obtener un artículo de la base de datos.

El concepto POST en cambio es enviar información desde el cliente paraa que sea procesada y actualice o agregue información en el servidor, como sería la carga o actualización en si de un artículo. Cuando enviamos (request) datos a través de un formulario, estos son procesados y luego a través de una redirección, ejemplo devolvemos (response) alguna página con información.

Tanto GET como POST solicitan una respuesta del servidor y ahí donde parecen que los conceptos son iguales ya que ambos logran los mismos objetivos.

Podríamos, aunque no es correcto, enviar por GET ciertos datos en una URL u actualizar o insertar dicha información en mi base de datos, pero realmente eso le corresponde al método POST. De la misma manera, solicitamos una página diferente por medio de POST y simplemente mostrarla como respuesta, aunque sería a través de una llamada GET.

Las llamadas por método GET pueden ser cacheadas (historial del navegador), indexadas por

buscadores, agregar los enlaces a nuestros favoritos o hasta pasar una URL completa a otra persona para que ingrese a dicha página. Con el método POST no se puede hacer esto.

Usamos links para ejecutar llamadas GET ya que la idea del link es "solicitar" una información (pagina) al servidor y que sea devuelta como una respuesta. Mientras usamos formularios para actualizar datos, como artículos, usuarios, etc. también en cuenta que el método POST envía más cantidad de datos que por GET.

8.2.8.1 Un caso de análisis

Suponga un carrito de compras, con una URL que agregue productos al carrito. Generalmente, la URL queda así (misitio.com/agregar_item.php?id=1). Al ser una llamada GET, Google podría indexar esa URL y aparecería en el buscador la palabra carrito. El problema aquí, si el visitante ejecuta esa página automáticamente agrega ese ítem al carrito de compras, lo cual no es la idea ya que el visitante al buscar el carrito de compras debe entrar al sitio y no agregar artículos ya que ni siquiera sabe cuál es. Por lo tanto, vemos que para este caso por más que usamos una URL, debemos usar el método POST, ejemplo:

```
1 <a onclick="f = document.createElement('form'); document.body.  
  appendChild(f);  
2 f.method = 'POST'; f.action = this.href; f.submit();return false;"  
3 href="agregar_item.php?id=1">Añadir al carrito  
4 </a>
```

Lo que hace este URL es muy sencillo, por medio del evento onclick de JavaScript, crea dinámicamente un formulario, le dice que es POST (ya que por defecto es GET), le asigna la URL del enlace al action del form, envía el formulario y retorna false para no ejecutar el link en sí. Para hacer esto usamos el helper link_to agregando la opción post=true:

```
<?php echo link_to('Añadir al carrito', 'agregar_item.php?id=1', 'post=true')?>
```

8.2.9 Enviando datos por enlace

Con el elemento HTML Ancla <a> también se envían datos (solamente) por GET :

```
<a href="/libros.php?autor=pepe&libro=cocoloco">Un libro interesante</a>
```

Cuando creamos un enlace en HTML, habitualmente se parece al código del siguiente cuadro (el lector lo puede transcribir en un archivo completo con la extensión.html):

```
<a href="destino.html">esto es un enlace que solicita al servidor el archivo llamado destino.html  
</a>
```

El objetivo de la etiqueta <a> es pedirle al servidor que entregue la página especificada en el atributo href que, en este caso, es la página destino.html. Obviamente, si no lo hemos ubicado ninguna página llamada destino.html dentro de la misma carpeta en la que está este archivo, nos dará un error cuando presionemos el enlace que solicita ese recurso.

Sin embargo, en las páginas dinámicas, aprovechamos la posibilidad de los enlaces para indicar al servidor, para que, además de que muestre una página en particular, envíe un valor a alguna variable disponible en el servidor, para que la utilice en el programa intérprete de PHP después, mientras procesa el código PHP de esa página, justo antes de que le devuelva el código HTML generado a nuestro navegador, que espera una respuesta.

La sintaxis para el envío de variables junto con un enlace es simple: definimos en el servidor una variable que llame nombre, cuyo valor es Pepe, y que el enlace vaya hacia una página llamada recibe.php.

Este enlace- que solicita unapaginany, a la vez, envía una variable con ese valor hacia el servidor – se expresa de la siguiente manera:

```
<a href="recibe.php?nombre=Pepe">
```

Solicitamos ver la pagina recibe.phpz de paso enviamos al servidor una variable llamada "nombre"conteniendo el valor "Pepe«/a>

Es importante observar que dentro del valor del atributo href, luego de especificar la URL solicitada (la página recibe.php), se ha agregado un signo de pregunta: ? Este signo indica que, a continuación, enviaremos una o más variables hacia el servidor web.

La sintaxis para enviar una variable consiste en colocar primero su nombre; luego, un signo igual;después, su valor de definición.

A diferencia de la sintaxis empleado en el lenguaje PHP, no debemos anteponer ningún signo "\$" al nombre de la variable, ni tampoco envolver entre comillas los valores de las variables, aun cuando sean alfanuméricos.

Esto es así ya que el envío de variables como parte de un enlace, no tiene relación con PHP ni con ningún otro lenguaje de programación en particular, sino que es una posibilidad que nos brinda el protocolo HTTP, que es el que comunica a un navegador con un servidor web. No son variables de PHP hasta que llegan al servidor y el programa interprete de PHP las lee.

En el ejemplo siguiente, creamos un archivo llamado enlaces.html y, otro, denominado destino.php.

El archivo enlaces.html es el siguiente:

```
1 <p>
2 <a href="destino.php?nombre=Pepe">Este es el enlace de Pepe</a><br />
3 <a href="destino.php?nombre=Pedro">Este es el enlace de Pedro</a><br
  />
4 <a href="destino.php?nombre="Juan"> Este es el enlace a Juan</a><br
  />
5 </p>
```

Una vez que el navegador visualiza esta página, según enlace presionado, enviamos hacia el servidor un valor distinto para la variable nombre. Siempre enviamos la misma variable, pero con distinta información en ella. Y sólo remitimos uno de los valores posibles (no existe un mouse que acepte simultáneamente más de un enlace).

El archivo destino.php da por sentado que llego esa variable, por esta razón, escribimos la orden print dentro del código HTML, que solicita al PHP que muestre el valor de la variable nombre.

Para eso, usamos esta sintaxis:

```
<php print ($_GET["nombre"]); ?>
```

Por supuesto, a esta página le falta la DTD y la apertura de las etiquetas html, head, body, etc., y sus correspondiente cierre, que lo agregamos al final.

8.2.10 Recuperar datos del formulario

Cuando un usuario presiona el botón enviar de un formulario, la información que contenían sus campos es enviada a una dirección URL desde donde tendremos que recuperarla para tratarla

de alguna manera. Por ejemplo, si realiza una compra, tendremos que recuperar los datos para completar el proceso de pago. La información del formulario "viaja" almacenada en variables que podremos recuperar y utilizar mediante PHP. Una de las formas de recuperación consiste en usar `$_POST`.

8.2.11 Recuperar variables POST

POST recupera datos enviados desde formularios con el método POST.

Escribe este código y guárdalo con un nombre de archivo como `ejemplo1.html`. A continuación, sube el archivo al servidor, introduce un nombre en el campo y visualiza el resultado.

```
1 <form name="formulario" method="post" action="ejemploPost.php">
2
3 Nombre: <input type="text" name="nombre" value="">
4
5 <input type="submit" />
6
7 </form>
```

Antes de presionar sobre el botón enviar, creamos el archivo `ejemploPost.php` que es el indicado en el formulario para el envío de los datos.

Escribe este código y guárdalo con un nombre de archivo como `ejemploPost.php`. A continuación, sube el archivo al servidor en la misma carpeta donde subiste el archivo php.

```
1 <?php
2 $nombre = $_POST['nombre'];
3 echo $nombre;
4 ?>
```

Una vez completemos el paso anterior, presionamos sobre el botón enviar del `ejemplo1.html` y vemos una imagen similar a la siguiente.

Vamos a explicar el proceso que ha tenido lugar.

El primer archivo es un documento HTML. Para ser más correctos, debemos escribirlo de esta manera:

```
1 <!DOCTYPE html>
2
3 <html>
4
5 <head>
6
7 <title>Ejemplo jodocha.com</title>
8
9 <meta charset="utf-8">
10
11 </head>
12
13 <body>
14
15 <form name="formulario" method="post" action="ejemploPost.php">
16
17 Nombre: <input type="text" name="nombre" value="">
18
```

```

19 <input type="submit" />
20
21 </form>
22
23 </body>
24
25 </html>

```

Sin embargo, comprobamos que los navegadores tratan de interpretar y mostrar el resultado de un código HTML (o PHP) incluso cuando la sintaxis o la definición del documento no es del todo correcta. Esto debemos conocerlo, sin embargo recomendamos que siempre se trate de ser lo más correctos posibles a la hora de escribir código web.

Vemos que hemos definido un formulario en cuya cabecera hemos puesto que el método de envío de los datos va a ser el método post y que la URL de destino va a ser `ejemploPost.php`. A su vez, el formulario tiene un campo cuyo atributo name es "nombre". Ese atributo define el nombre de la variable que vamos a recuperar en la URL de destino.

En el archivo php hemos incluido la línea `$nombre = $_POST['nombre'];` que significa que creamos una variable php denominada `$nombre` donde almacenamos la información del campo 'nombre' a través del método post proveniente del formulario. Si tuviéramos otros campos que hubiéramos definido como apellidos, teléfono, edad, el tratamiento sería similar. Por ejemplo:

```

1
2 $apellidos = $_POST['apellidos'];
3
4 $telefono = $_POST['celular'];
5
6 $edadPersona = $_POST['edad'];

```

Fíjate en que una cosa es la variable en la que almacenamos la información recuperada, y otra cosa es el nombre del campo del formulario de donde proviene. Por ejemplo en `$apellidos = $_POST['apellidos'];` coinciden el nombre de la variable que utilizamos con el nombre del campo del formulario. Sin embargo, en `$telefono = $_POST['celular'];` no coinciden. En este caso, el campo que proviene del formulario se llama 'celular' mientras que la información que venga en ese campo la almacenamos en una variable a la que hemos llamado `$telefono`. Finalmente, en `$edadPersona = $_POST['edad'];` estamos almacenando en una variable a la que hemos llamado `$edadPersona` la información proveniente de un campo del formulario denominado 'edad'.

Con frecuencia los nombres de las variables y de los campos del formulario se hacen coincidir, pero en otras ocasiones no. Esto queda a elección del programador.

Como podemos observar, recuperar datos enviados por un formulario con el método POST es bastante simple usando PHP.

8.2.12 Recuperar datos de formulario REQUEST

Cuando un usuario presiona el botón enviar de un formulario, la información que contenían sus campos es enviada a una dirección URL desde donde tendremos que recuperarla para tratarla de alguna manera. Por ejemplo, si realiza una compra, tendremos que recuperar los datos para completar el proceso de pago. La información del formulario "viaja" almacenada en variables que recuperamos y utilizamos mediante PHP. Una de las formas de recuperación consiste en usar `$_REQUEST`.

8.2.13 Recuperar variables con REQUEST

REQUEST nos permite capturar variables enviadas desde formularios con los métodos GET o POST. Vamos a ver dos ejemplos de formularios (ejemplo1.html y ejemplo2.html), que en un caso se enviarán usando GET y en otro usando POST. Ambos formularios enviarán la información (action) a una página común desde donde recuperaremos los datos usando `$_REQUEST`.

El código de los archivos html sería el siguiente para ejemplo1.html y ejemplo2.html. Escribe el código en un editor de texto como Notepad++ y visualízalos en tu navegador:

```
1 <form name="formulario" method="get" action="ejemploRequest.php">
2
3 Nombre: <input type="text" name="nombre" value="">
4
5 <input type="submit" />
6
7 </form>
8
9 <form name="formulario" method="post" action="ejemploRequest.php">
10
11 Nombre: <input type="text" name="nombre" value="">
12
13 <input type="submit" />
14
15 </form>
```

Como observamos, el ejemplo1.html envía los datos por GET mientras que el ejemplo2.html envía los datos por POST. Ahora bien, la acción o destino donde se enviarán los datos es la misma en los dos casos, la dirección ejemploRequest.php.

Escribe este código y guárdalo con un nombre de archivo como ejemploRequest.php. A continuación, sube el archivo al servidor en la misma carpeta donde subiste el ejemplo1.html y ejemplo2.html

```
1 <?php //Ejemplo jodocha.com
2
3 $nombre = $_REQUEST['nombre'];
4
5 echo $nombre;
6
7 ?>
```

A continuación, el resultado obtenido al introducir el nombre tanto en el ejemplo1.html como en el ejemplo2.html, y ve que es el mismo.

Vamos a explicar el proceso que ha tenido lugar. El primer archivo es un documento HTML. Para ser más correctos, deberíamos haberlo escrito de esta manera:

```
1 <!DOCTYPE html>
2
3 <html>
4
5 <head>
6
7 <title>Ejemplo jodocha.com</title>
8
```

```

9  <meta charset="utf-8">
10
11 </head>
12
13 <body>
14
15 <form name="formulario" method="get" action="ejemploRequest.php">
16
17 Nombre: <input type="text" name="nombre" value="">
18
19 <input type="submit" />
20
21 </form>
22
23 </body>
24
25 </html>

```

Sin embargo, comprobamos que los navegadores tratan de interpretar y mostrar el resultado de un código HTML (o PHP) incluso cuando la sintaxis o la definición del documento no es del todo correcta. Esto debemos conocerlo, sin embargo recomendamos que siempre se trate de ser lo más correctos posibles a la hora de escribir código web.

Vemos que hemos definido un formulario en cuya cabecera hemos puesto que el método de envío de los datos va a ser el método GET y que la URL de destino va a ser ejemploRequest.php. A su vez, el formulario tiene un campo cuyo atributo name es "nombre". Ese atributo define el nombre de la variable que vamos a recuperar en la URL de destino. En el archivo php incluimos la línea `$nombre = $_REQUEST['nombre'];` que significa que creamos una variable php denominada `$nombre` donde almacenamos la información del campo 'nombre' que recibe la URL a través del método GET proveniente del formulario. Si tuviéramos otros campos que hubiéramos definido como apellidos, teléfono, edad, el tratamiento es similar. Por ejemplo:

```

1  $apellidos = $_REQUEST['apellidos'];
2
3  $telefono = $_REQUEST['celular'];
4
5  $edadPersona = $_REQUEST['edad'];

```

Fíjate en que una cosa es la variable en la que almacenamos la información recuperada, y otra cosa es el nombre del campo del formulario de donde proviene. Por ejemplo en `$apellidos = $_REQUEST['apellidos'];` coinciden el nombre de la variable que utilizamos con el nombre del campo del formulario. Sin embargo, en `$telefono = $_REQUEST['celular'];` no coinciden. En este caso, el campo que proviene del formulario se llama 'celular' mientras que la información que venga en ese campo la almacenamos en una variable a la que hemos llamado `$telefono`. Finalmente, en `$edadPersona = $_REQUEST['edad'];` estamos almacenando en una variable a la que hemos llamado `$edadPersona` la información proveniente de un campo del formulario denominado 'edad'. Con frecuencia los nombres de las variables y de los campos del formulario se hacen coincidir, pero en otras ocasiones no. Esto queda a elección del programador.

8.2.14 Ejercicio Resuelto

Diseñar un formulario web que pida la altura y el diámetro de un cilindro en metros. Una vez el usuario introduzca los datos y pulse el botón calcular, calcule el volumen del cilindro y muestre el resultado en el navegador. El envío de datos debe hacerse por GET y la recuperación con REQUEST.

Solución

La solución esquematizada en pseudocódigo es la siguiente:

1. Inicio
2. Mostrar "Introduzca el diámetro, en metros": Pedir D
3. Mostrar "Introduzca la altura, en metros": Pedir H
4. $R = D/2$: $Pi = 3,141593$
5. $V = Pi * (R ^ 2) * H$
6. Mostrar "El volumen del cilindro es de", V, "metros cúbicos"
7. Fin

Esquematizar la solución en pseudocódigo es una buena idea antes de realizar la programación, pues nos permite definir conceptualmente cómo va a ser nuestro código antes de escribirlo. Es sobre todo adecuado para personas que se están iniciando en la programación. Crearemos el archivo html:

```
1  <!DOCTYPE html>
2
3  <html>
4
5  <head>
6
7  <title>Ejemplo jodocha.com</title>
8
9  <meta charset="utf-8">
10
11 </head>
12
13 <body>
14
15 <form name="formularioDatos" method="get" action="ejemploRequest1.php
    ">
16
17 <p> CÁLCULO DEL VOLUMEN DE UN CILINDRO </p>
18
19 <br/>
20
21 Introduzca el diámetro en metros: <input type="text" name="diam"
    value="">
22
23 <br/> <br/>
```



```

24
25 Introduzca la altura en metros: <input type="text" name="altu" value
    ="">
26
27 <br/> <br/>
28
29 <input value="Calcular" type="submit" />
30
31 </form>
32
33 </body>
34
35 </html>

```

Por otro lado, creamos el archivo php con el tratamiento de datos:

```

1 <?php //Ejemplo jodocha.com
2
3 $diametro = $_REQUEST['diam'];
4
5 $altura = $_REQUEST['altu'];
6
7 $radio = $diametro/2;
8
9 $Pi = 3.141593;
10
11 $volumen = $Pi*$radio*$radio*$altura;
12
13 echo "<br/> &nbsp; ; El volumen del cilindro es de". $volumen. "metros
    cúbicos";
14
15 ?>

```

Fíjate que hemos escrito la potencia del radio como `$radio * $radio`. En otros lenguajes existe el operador de exponenciación, pero en php esta operación se tiene que realizar recurriendo a una función matemática. Esta función la estudiaremos en otro momento.

Finalmente obtenemos un resultado. Haz pruebas introduciendo como valores de diámetro y altura 2,15 y 1,75 en vez de 2.15 y 1.75. Posiblemente no obtengas un resultado adecuado si usas las comas, ya que en PHP el separador de la parte decimal de un número es el punto.

8.2.14.1 Ejercicio resuelto

Diseñar un desarrollo web simple con php que pida al usuario el precio de tres productos en tres establecimientos distintos denominados "Tienda 1", "Tienda 2" y "Tienda 3". Una vez se introduzca esta información se debe calcular y mostrar el precio medio del producto. El envío de datos debe hacerse por POST y la recuperación con REQUEST.

SOLUCIÓN

La solución esquematizada en pseudocódigo es la siguiente:

1. Inicio

2. Mostrar "Introduzca el precio del producto en el establecimiento número 1, en Bolívares":
Pedir Precio1
3. Mostrar "Introduzca el precio del producto en el establecimiento número 2, en Bolívares":
Pedir Precio2
4. Mostrar "Introduzca el precio del producto en el establecimiento número 3, en Bolívares":
Pedir Precio3
5. $Media = (Precio1 + Precio2 + Precio3) / 3$
6. Mostrar "El precio medio del producto es", Media, "Bolívares"
7. Fin

```
1 <!DOCTYPE html>
2
3 <html>
4
5 <head>
6
7 <title>Ejemplo jodocha.com</title>
8
9 <meta charset="utf-8">
10
11 </head>
12
13 <body>
14
15 <form name="formularioDatos" method="post" action="ejemploRequest2.
    php">
16
17 <p> CÁLCULO DEL PRECIO MEDIO DE UN PRODUCTO </p>
18
19 <br/>
20
21 Introduzca el precio del producto en el establecimiento número 1, en
    Bolívares: <input type="text" name="precio1" value="">
22
23 <br/> <br/>
24
25 Introduzca el precio del producto en el establecimiento número 1, en
    Bolívares: <input type="text" name="precio2" value="">
26
27 <br/> <br/>
28
29 Introduzca el precio del producto en el establecimiento número 3, en
    Bolívares: <input type="text" name="precio3" value="">
30
31 <br/> <br/>
32
33 <input value="Calcular" type="submit" />
34
```

```

35 </form>
36
37 </body>
38
39 </html>

```

Es posible que durante el desarrollo del curso te encuentres visualizaciones de este tipo, donde podrás comprobar que los acentos o tildes no se ven bien.

Recordarte que la solución a esto es, cuando estamos trabajando con Notepad++, elegir en el menú Formato la opción `Codificar en UTF-8 sin BOM`. En caso de que por error el archivo esté en otro formato, elige la opción `Convertir en UTF-8 sin BOM` para dejar correctamente la codificación del archivo. También puede ser necesario introducir la etiqueta: `<meta charset=utf-8>` entre las etiquetas `<head>... </head>` para lograr que la visualización sea correcta.

Por otro lado, necesitamos el archivo php.

```

1 <?php //Ejemplo jodocha.com
2
3 $precio1 = $_REQUEST['precio1'];
4
5 $precio2 = $_REQUEST ['precio2'];
6
7 $precio3 = $_REQUEST ['precio3'];
8
9 $media = ($precio1+$precio2+$precio3)/3;
10
11 echo "<br/> &nbsp; DATOS RECIBIDOS";
12
13 echo "<br/> &nbsp; ; Precio producto establecimiento 1: ". $precio1. "
    Bolívares";
14
15 echo "<br/> &nbsp; ; Precio producto establecimiento 2: ". $precio2. "
    Bolívares";
16
17 echo "<br/> &nbsp; ; Precio producto establecimiento 2: ". $precio3. "
    Bolívares <br/>";
18
19 echo "<br/> &nbsp; ; El precio medio del producto es de ". $media. "
    Bolívares";
20
21 ?>

```

8.3 PHP con MySQL

Este capítulo no pretende ser un manual de MySQL, pero sí explicamos como realizar consultas básicas desde PHP.

Trabajamos algunas funciones básicas para acceder a una base de datos MySQL desde PHP para recuperar, insertar o borrar información.

No vamos a estudiar en profundidad las bases de datos ni el lenguaje SQL. Tratamos de ser prácticos, conocer algunas funciones para acceder a bases de datos con PHP y aprender a usarlas a través de ejemplos.

Mostramos el acceso a base de datos con PHP de dos formas distintas: las funciones tipo mysql, como mysql_connect, son antiguas, cuyo uso se desaconseja si está trabajando con un servidor con una versión de MySQL moderna. Las funciones tipo mysqli, como mysqli_connect, son modernas, cuyo uso se recomienda siempre que trabajes con servidores actualizados. Las funciones mysqli se escriben de dos formas: con estilo orientado a objetos o por procedimientos. Veamos cómo usar estas funciones con el estilo por procedimientos, que es más sencillo inicialmente.

8.3.1 Función mysqli_connect

Conexión Previo a la conexión es buena idea almacenar la información necesaria en constantes que faciliten su posterior modificación.

```
define("DB_HOST","localhost");
define("DB_USER","usuario");
define("DB_PASS","mypassword");
define("DB_DATABASE","Demo");
```

1. Interfaz orientada a objetos

Para la conexión simplemente hay que crear una instancia de mysqli pasando los datos necesarios para la conexión: host, usuario, contraseña y nombre de la base de datos.

```
1 | $mysqli = new mysqli(DB_HOST, DB_USER, DB_PASS, DB_DATABASE);
2 | if($mysqli->connect_errno > 0){
3 |     die("Imposible conectarse con la base de datos [" . $mysqli->
    |         connect_error . "]);
4 | }
```

También es posible dejar sin indicar la base de datos al crear la instancia y hacerlo posteriormente mediante `$mysqli -> select_db()`. Este método devuelve un bool para verificar el posible error al seleccionar la base de datos.

```
1 | $mysqli = new mysqli(DB_HOST, DB_USER, DB_PASS);
2 | if(!$mysqli->select_db(DB_DATABASE)) die ($mysqli->error);
```

2. Interfaz procedimental

Creamos la conexión mediante el procedimiento mysqli_connect() que tiene como parámetros los datos necesarios para la conexión y devuelve un objeto que representa la conexión al servidor mysql.

```
1 | $con = mysqli_connect(DB_HOST, DB_USER, DB_PASS, DB_DATABASE);
2 | if (mysqli_connect_errno()) {
3 |     echo "Imposible conectarse a la base de datos: " .
    |         mysqli_connect_error();
4 | } else {
5 | }
```

3. Interfaz procedimental

Hemos visto a la hora de conectar con la base de datos o de mostrar errores, todo método y propiedades disponibles en la interfaz orientada a objetos tiene un equivalente en la interfaz

procedimental.

El equivalente para el método `query()` de la interfaz orientada a objetos lo tenemos en el procedimiento `mysqli_query(mysqli $link, string $consulta)`. Ahora, aparte de la cadena que contiene la consulta, también necesita un identificador devuelto por `mysqli_connect()`. Otro método importante es `mysqli_affected_rows(mysqli $link)` que devuelve el número de filas afectadas por la consulta `INSERT`, `UPDATE`, o `DELETE` ejecutada anteriormente.

- Ejemplo de creación de una tabla

```
1  $con = mysqli_connect(DB_HOST, DB_USER, DB_PASS,
    DB_DATABASE);
2  ...
3  $sql = "CREATE TABLE usuario ( ID INT NOT NULL PRIMARY KEY
    AUTO_INCREMENT,
4  nombre VARCHAR(50) NOT NULL, mail VARCHAR(100) NOT NULL)
    ENGINE=InnoDB";
5  $resultado = mysqli_query($con, $sql);
6  if(mysqli_errno($con)) die(mysqli_error($con));
```

- Ejemplo de inserción de una fila

```
1  $con = mysqli_connect(DB_HOST, DB_USER, DB_PASS,
    DB_DATABASE);
2  ...
3  $sql = "INSERT INTO usuario (nombre, mail) VALUES ('usuario
    1', 'usuario1@ejemplo.com')";
4  $resultado = mysqli_query($con, $sql);
5  if(mysqli_errno($con)) die(mysqli_error($con));
```

Para obtener el ID de la última inserción `mysqli` proporciona el procedimiento `mysqli_insert_id(mysqli $link)`.

```
1  echo "El id de la última inserción es " . mysqli_insert_id(
    $con);
```

- Ejemplo de actualización de una fila

```
1  $sql = "UPDATE usuario SET name='jose' WHERE id = '2'";
2  $resultado = mysqli_query($con, $sql);
3  if(mysqli_errno($con)) die(mysqli_error($con));
```

- Ejemplo de borrado de tabla (si existe)

```
1  $con = mysqli_connect(DB_HOST, DB_USER, DB_PASS,
    DB_DATABASE);
2  ...
3  $sql = "DROP TABLE IF EXISTS usuario";
4  $resultado = mysqli_query($con, $sql);
5  if(mysqli_errno($con)) die(mysqli_error($con));
```

- Ejemplo de borrado de una fila de la tabla

```
1  $con = mysqli_connect(DB_HOST, DB_USER, DB_PASS,
    DB_DATABASE);
2  ...
```

```

3 | $sql = "DELETE FROM usuario WHERE id = '1'";
4 | $resultado = mysqli_query($con, $sql);
5 | if(mysqli_errno($con)) die(mysqli_error($con));

```

También, buscamos saber si la consulta no ha se ha ejecutado con el resultado esperado. Para ello consultamos el número de filas afectadas por la consulta.

```

1 | if(mysqli_affected_rows($con)!=1) die ('Error ...');

```

8.3.2 Función `mysqli_close`

Esta función cierra la conexión con una base de datos anteriormente abierta. Es recomendable cerrar una conexión una vez que hayamos terminado de usarla.

La sintaxis que usamos es:

`mysqli_close(\$nombreConexión);` Para servidores no actualizados usamos `mysql_close` en lugar de `mysqli_close`

La función devuelve TRUE si se ha cerrado correctamente o FALSE en caso de error.

El identificador o nombre de conexión es aquel obtenido previamente de la función `mysqli_connect`.

8.4 Consultas que no devuelven resultado

La ejecución de consultas tipo "DROP", "DELETE", "CREATE", "UPDATE" e "INSERT" devuelve true o false en caso de error. Pero no devuelven algún tipo de dato almacenado en la base de datos.

1. Interfaz orientada a objetos

Para este tipo de consultas utilizamos el método `query()` de la interfaz orientada a objetos. Dicho método sólo necesita un parámetro que es la cadena que contiene la consulta.

Una propiedad importante, si usamos esta interfaz, es `$mysqli->affected_rows` que devuelve el número de filas afectadas por la última consulta INSERT, UPDATE, o DELETE ejecutada.

- Ejemplo de creación de una tabla.

Para este caso utilizamos cadenas en PHP.

```

1 | $sql = "CREATE TABLE usuario ( ID INT NOT NULL PRIMARY
   |     KEY AUTO_INCREMENT,
2 |     nombre VARCHAR(50) NOT NULL, mail VARCHAR(100) NOT NULL)
   |     ENGINE=InnoDB ";
3 | $mysqli->query($sql);
4 | if($mysqli->errno) die($mysqli->error);

```

- Ejemplo de inserción de una fila.

```

1 | $sql = "INSERT INTO usuario (nombre, mail) VALUES ('
   |     usuario 1', 'usuario1@ejemplo.com')";

```

```

2 | $mysqli->query($sql);
3 | if($mysqli->errno) die($mysqli->error);

```

Los ID's de las tablas pueden ser autoincrementales por lo que mysql se encarga de ello y no tenemos que definirlo a la hora de la inserción. Pero puede ser útil conocer el ID de la última inserción, y para ello mysqli proporciona la propiedad.

```

1 | $mysqli->insert_id.

```

- Ejemplo de actualización de una fila

```

1 | $sql = "UPDATE usuario SET name='jose' WHERE id = '2'";
2 | $mysqli->query($sql);
3 | if($mysqli->errno) die($mysqli->error);

```

- Ejemplo de borrado de tabla Llevamos a cabo una comprobación de la existencia de la tabla para que devuelva un posible error.

```

1 | $sql = "DROP TABLE IF EXISTS usuario";
2 | $mysqli->query($sql);
3 | if($mysqli->errno) die($mysqli->error);

```

También buscamos saber si la consulta ha provocado un resultado esperado, comprobando el número de filas afectadas por la misma. Ya que puede que no haya error de sintaxis en la consulta (error en mysql), pero que el resultado sea erróneo.

```

1 | if($mysqli->affected_rows!=1) die ('Error...');

```

- Ejemplo de borrado de una fila de la tabla

```

1 | $sql = "DELETE FROM usuario WHERE id = '1'";
2 | $mysqli->query($sql);
3 | if($mysqli->errno) die($mysqli->error);

```

8.5 Recuperar resultados: DATA__SEEK, FETCH__ARRAY (MYSQL__RESULT)

Una sentencia de consulta normalmente devuelve "un conjunto de resultados" que según el ejemplo de sintaxis anterior tenemos una variable denominada \$result. Veamos los resultados de una consulta:

nombre	apellidos	direccion	telefono	edad	altura
Manuel Jesús	López de la Rosa	C/Juan Bautista Nº 3	658954875	32	1.80
María	Manzano Cabezas	C/Arco del triunfo Nº 7	695001002	19	1.99

La pregunta es: ¿Cómo extraer el dato de una celda concreta? ¿el nombre de la primera fila? Para ello, escribimos: `mysqli_data_seek ($result, numeroDeFila);` o también `$result->data_seek (numeroDeFila);`
`$extraido= mysqli_fetch_array($result);` o también válido `$extraido=$result->fetch_array()`
`;`

Estamos dando dos pasos: una sentencia como `mysqli_data_seek ($result, 0);` significa "posicionarte en la fila 0 de los resultados"(la primera fila). También es válido escribir `$result->data_seek(0);` que tiene el mismo efecto. Si escribimos `$result->data_seek(15);` significa "posicionarte en la fila 16 de los resultados"(tener en cuenta que se empieza a contar por cero, de ahí que 0, 1, 2, ... , 15 resulten 16 filas).

La sentencia `$extraido= mysqli_fetch_array($result);` indica que los valores existentes en la fila se introduzcan en un array cuyos índices en principio pueden ser tanto asociativos (el nombre de la columna) como numéricos (empezando por cero). Por ejemplo si la primera columna en la tabla de la base de datos es «ciudad» podemos usar `$extraido['ciudad']` para acceder al valor existente en la fila con la que estemos trabajando para la columna ciudad. Igualmente podríamos usar `$extraido[0]` para referirnos a la primera columna, `$extraido[1]` para la segunda columna, `$extraido[2]` para la tercera columna y así sucesivamente.

8.5.1 `mysqli_result`

En servidores no actualizados se puede usar la función `mysqli_result` aunque al no pertenecer a las funciones `mysqli` no está recomendado su uso.

La sintaxis es: `mysqli_result($result, $numeroDeFila, $identificadorDeLaColumna);`

Esta función devuelve el contenido de la celda en la fila y columna indicadas en forma de string (cadena de caracteres) en caso de éxito, o `FALSE` en caso de error.

`$result` es el resultado obtenido previamente con una invocación a la función `mysqli_query`.

`numeroDeFila` es un número de fila dentro del resultado obtenido teniendo en cuenta que los números de fila se cuentan empezando desde cero.

`identificadorDeLaColumna` puede ser el índice del campo (por ejemplo 0), el nombre del campo (por ejemplo ciudad), o el nombre de la tabla punto nombre del campo (por ejemplo agenda.ciudad)

Finalmente, se puede especificar el nombre del campo que queremos extraer dentro de la fila (esto es opcional).

8.5.2 Ejemplo de las funciones anteriores

Escribe este código en un editor y guárdalo en un archivo de nombre `ejemplo1.php` (recuerda que debes crear la base de datos y los datos de ejemplo). Recuerda cambiar y poner tus propios valores en las funciones `mysqli_connect` y `mysqli_select_db`.

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 </head>
5 <body>
6 <?php
7
8 $link = mysqli_connect("sql203.byethost7.com", "b7_10356956",
   "*****");
9
10 mysqli_select_db($link, "b7_10356956_mibasededatos");
11
```



```

12 $tildes = $link->query("SET NAMES 'utf8'"); //Para que se muestren
    las tildes
13
14 $result = mysqli_query($link, "SELECT * FROM agenda");
15
16 mysqli_data_seek ($result, 0);
17
18 $extraido= mysqli_fetch_array($result);
19
20 echo "- Nombre: ".$extraido['nombre']."<br/>";
21
22 echo "- Apellidos: ".$extraido['apellidos']."<br/>";
23
24 echo "- Dirección: ".$extraido['direccion']."<br/>";
25
26 echo "- Teléfono: ".$extraido['telefono']."<br/>";
27
28 echo "- Edad: ".$extraido['edad']."<br/>";
29
30 mysqli_free_result($result);
31
32 mysqli_close($link);
33
34 ?>
35
36 </body>
37
38 </html>

```

El código con mysql (no recomendado) es:

```

1 <html>
2 <body>
3 <?php
4
5 $link = mysql_connect("sql203.byethost7.com", "b7_10356956", "*****")
    ;
6
7 mysql_select_db("b7_10356956_mibasededatos", $link);
8
9 mysql_query("SET NAMES 'utf8'"); //Para que se muestren las tildes
10
11 $result = mysql_query("SELECT * FROM agenda", $link);
12
13 echo"Nombre: ".mysql_result($result, 0, "nombre")."<br>";
14
15 echo"Apellidos: ".mysql_result($result, 0, "apellidos")."<br>";
16
17 echo"Dirección: ".mysql_result($result, 0, "direccion")."<br>";
18
19 echo"Teléfono:".mysql_result($result, 0, "telefono")."<br>";
20
21 echo"Edad:".mysql_result($result, 0, "edad")."<br>";
22

```

```

23 echo "Altura:".mysql_result($result, 0, "altura")."<br>";
24
25 mysql_free_result($result);
26
27 mysql_close($link);
28
29 ?>
30 </body>
31 </html>

```

En este código usamos las funciones y la base de datos y tablas que creados anteriormente. Con ello realizamos una consulta a la base de datos y mostramos los resultados por pantalla.

Interpretemos las líneas escritas: `$link = mysqli_connect("sql203.byethost7.com", "b7_10356956", "*****");`

Con esta línea introducimos el identificador de conexión en una variable denominada `$link`. Invocamos la función `mysqli_connect` pasando como parámetros el nombre del hosting, el nombre de usuario y la contraseña. Estos datos debemos conocerlos (o consultarlos) previamente. `mysqli_select_db($link, "b7_10356956_mibasededatos");`

Con esta línea seleccionamos la base de datos pasando como parámetros el identificador de conexión (obtenido y guardado en una variable) y el nombre de base de datos. `$result = mysqli_query($link, "SELECT * FROM agenda");`

Con esta línea seleccionamos todos los registros (filas) y campos (columnas) existentes en la tabla "agenda". Para seleccionar todos los registros escribimos `SELECT *`. A continuación, indicamos la tabla de la cual queremos dichos registros y lo expresamos con la sintaxis `FROM agenda`, siendo `agenda` el nombre de la tabla en la base de datos. En la llamada a la función incluimos el identificador de conexión obtenido previamente.

Con la sentencia `mysqli_data_seek ($result, 0);` vamos a la primera fila (fila cero) de los resultados de la consulta.

Con la sentencia `extraido= mysqli_fetch_array($result);` introducimos en un array denominado `$extraido` los datos de la fila cero obtenidos de la consulta.

Finalmente, mostramos por pantalla los resultados.

En la versión que no usa `mysqli` (no recomendada) con la sentencia `mysql_result($result, 0, "nombre")` obtenemos el campo "nombre" del primer registro (registro 0) del resultado de la consulta almacenado previamente en `$result`.

El resultado de invocar el archivo creado muestra en pantalla el nombre, apellidos, dirección, teléfono, edad y altura almacenados en la primera fila de nuestra base de datos. Algo similar a esto: Resumiendo lo que hemos hecho, hemos usado la función `mysqli_connect()`, que abre una conexión con el servidor MySQL en el Host especificado (en este ejemplo la máquina donde está alojada el servidor MySQL es `sql203.byethost7.com`, pero tú tendrás que introducir tu propio dato). También hemos especificado un usuario (`b7_10356956` en este ejemplo, pero tú tendrás que introducir tu propio dato), y un password para el usuario indicado (tendrás que escribir tu password de usuario de base de datos).

Si la conexión ha tenido éxito, la función `mysqli_connect()` devuelve un identificador de dicha conexión que es almacenado en la variable `$link`. Si no tuviera éxito, devuelve 0 (FALSE).

Con `mysqli_select_db()` PHP solicitamos al servidor que en la conexión `$link` nos queremos conectar a la base de datos indicada.

La siguiente función `mysqli_query()`, usando el identificador de la conexión (`$link`), envía una instrucción SQL al servidor MySQL para que éste la procese. El resultado de ésta operación es almacenado en la variable `$result`.

Finalmente, elegimos la fila de resultados cero con `mysqli_data_seek ($result, 0)`; y guardamos los resultados de esa fila en un array con `$extraido= mysqli_fetch_array($result)`; Si quisiéramos mostrar los siguientes registros tendríamos que incluir instrucciones con los números 1, 2, ...

La sentencia `mysqli_free_result($result)`; libera la conexión establecida con la base de datos.

La sentencia `mysqli_close($link)`; cierra la conexión con la base de datos.

Nota: Si tenemos algún problema en localizar la información de nuestro proveedor (nombre de base de datos, usuario, etc.). recordar que dicha información normalmente está disponible en el panel de control del servidor.

8.5.3 Ejercicio propuesto

Utilizando la tabla `ciudades` cuyo contenido es:

id	ciudad	pais	habitantes	superficie	tieneMetro
1	Ciudad de México	México	555666	23434.34	1
2	Barcelona	España	444333	1111.11	0
3	Buenos Aires	Argentina	888111	333.33	1
4	Medellín	Colombia	999222	888.88	0
5	Lima	Perú	999111	222.22	0
6	Caracas	Venezuela	111222	111.11	1
7	Santiago	Chile	777666	222.22	1
8	Antigua	Guatemala	444222	877.33	0
9	Quito	Ecuador	333111	999.11	1
10	La Habana	Cuba	111222	333.11	0

Crea un archivo php para realizar una consulta a la tabla y muestre en pantalla los valores de país y ciudad de cada fila.

Los resultados de la consulta a la tabla son los siguientes:

País	Ciudad
México	Ciudad de México
Venezuela	Caracas
España	Madrid
Argentina	Buenos Aires
Cuba	La Habana

8.6 Formulario para cargar datos

Comencemos insertando datos. Para ello, necesitamos un formulario HTML.

```

1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <title>Registrar persona</title>
6 </head>
7 <body>
8 <form method="post" action="nuevaPersona.php">
9 <label for="nombre">Nombre:</label>
10 <br>
11 <input name="nombre" required type="text" id="nombre" placeholder="
    Escribe tu nombre...">
12 <br><br>
13 <label for="apellidos">Apellidos:</label>
14 <br>
15 <input name="apellidos" required type="text" id="apellidos"
    placeholder="Escribe tus apellidos...">
16 <br><br>
17 <label for="sexo">Género</label>
18 <select name="sexo" required name="sexo" id="sexo">
19 <option value="">--Selecciona--</option>
20 <option value="M">Masculino</option>
21 <option value="F">Femenino</option>
22 </select>
23 <br><br><input type="submit" value="Registrar">
24 </form>
25 </body>
26 </html>

```

Atentos al atributo action, en donde ponemos el nombre del archivo php que recibe los datos.

Ese archivo debe estar en la misma carpeta que el formulario. Si cambiamos nuevaPersona.php de ubicación o nombre, basta con cambiarlo en el formulario para que las cosas sigan funcionando.

Si movemos el archivo nuevaPersona.php a la carpeta "foo"(donde está el formulario), cambiamos el action, así: action="foo/nuevaPersona.php"

Si lo cambio a una carpeta arriba, puedo ponerlo así:

action="../carpeta/nuevaPersona.php"

Y puedo tener miles de carpetas:

action="una/carpeta/otra/otra/otroNombre.php"

8.6.1 Recibiendo datos del formulario

Antes de recibirlos, por favor observa el atributo "name"de cada input. Es el nombre que usaremos para acceder a los datos, distinguiendo mayúsculas y minúsculas. Veamos ahora el archivo nuevaPersona.php, donde recibiremos los datos introducidos en el formulario.

```

1 <?php
2 #Salir si alguno de los datos no está presente
3 if(!isset($_POST["nombre"]) || !isset($_POST["apellidos"]) || !isset(
    $_POST["sexo"])) exit();
4 #Si todo va bien, se ejecuta esta parte del código...
5 include_once "base_de_datos.php";

```

```

6 $nombre = $_POST["nombre"];
7 $apellidos = $_POST["apellidos"];
8 $sexo = $_POST["sexo"];
9 /*
10 Al incluir el archivo "base_de_datos.php", todas sus variables están
11 a nuestra disposición. Por lo que podemos acceder a ellas tal como si
    hubiéramos
12 copiado y pegado el código
13 */
14 $sentencia = $base_de_datos->prepare("INSERT INTO personas(nombre,
    apellidos, sexo) VALUES (?, ?, ?);");
15 $resultado = $sentencia->execute([$nombre, $apellidos, $sexo]); #
    Pasar en el mismo orden de los ?
16 #execute regresa un booleano. True en caso de que todo vaya bien,
    falso en caso contrario.
17 #Con eso podemos evaluar
18 if($resultado === TRUE) echo "Insertado correctamente";
19 else echo "Algo salió mal. Por favor verifica que la tabla exista";
20 ?>

```

Si todo va bien, al visitar el formulario y enviarlo muestra un "Insertado correctamente".

8.6.2 Listar datos estáticos

Una vez que insertamos, mostramos las personas en la tabla. Para ello, usaremos una tabla. Antes de confundir al lector con la creación de la tabla, veamos una de ellas en su forma estática. Así luce el código:

```

1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <title>Tabla de ejemplo</title>
6 <style>
7 table, th, td {
8 border: 1px solid black;
9 }
10 </style>
11 </head>
12 <body>
13 <table>
14 <thead>
15 <tr>
16 <th>ID</th>
17 <th>Nombre</th>
18 <th>Apellidos</th>
19 <th>Género</th>
20 </tr>
21 </thead>
22 <tbody>
23 <tr>
24 <td>1</td>
25 <td>John</td>
26 <td>Doe</td>

```

```

27 <td>M</td>
28 </tr>
29 <tr>
30 <td>2</td>
31 <td>María José</td>
32 <td>Ejemplo</td>
33 <td>F</td>
34 </tr>
35 <tr>
36 <td>3</td>
37 <td>Pedro</td>
38 <td>Ramírez</td>
39 <td>M</td>
40 </tr>
41 </tbody>
42 </table>
43 </body>
44 </html>

```

Si observas bien, ves que la estructura se conserva y sólo cambia en tr. Es decir, se repite la parte del código en donde dice `<tr><td>`.

Pero eso lo puede hacer cualquiera, y además, ¿qué beneficio trae, si tenemos que hacerla a mano? mejor debemos hacer que PHP la genere por nosotros, o bueno, darle sólo una pequeña ayuda. Veamos entonces cómo hacer eso.

8.6.3 Listar datos dinámicos

Ahora vemos cómo listar datos de MySQL. Para ello, utilizamos código PHP. El archivo que lista los datos, al menos en su primer versión, se ve así:

```

1 <?php
2 include_once "base_de_datos.php";
3 $sentencia = $base_de_datos->query("SELECT * FROM personas;");
4 $personas = $sentencia->fetchAll(PDO::FETCH_OBJ);
5 print_r($personas);
6 ?>
7
8 <?php
9 include_once "base_de_datos.php";
10 $sentencia = $base_de_datos->query("SELECT * FROM personas;");
11 $personas = $sentencia->fetchAll(PDO::FETCH_OBJ);
12 print_r($personas);
13 ?>

```

Nota: El parámetro que pasamos a `fetchAll` es una constante estática pública de la clase PDO, (`PDO::FETCH_OBJ`) y accede a las filas de la tabla como si fuera un objeto nativo de PHP. Es decir, accedemos al nombre de la persona usando `$persona->nombre` en lugar de (es la opción que está por defecto) `$persona[1]`.

Recordemos que el código, entre más expresivo, mejor.

Bueno, con todo esto tenemos un arreglo que se ve así al visitar la página:

Esto es porque sólo he registrado a una persona. Registre más personas.

Pero sigue sin verse bonito, aquí es en donde combinamos HTML y PHP para convertir ese arreglo en una tabla bonita, o al menos mejor que ese arreglo no entendible. El código queda así:

```
1  <?php
2  include_once "base_de_datos.php";
3  $sentencia = $base_de_datos->query("SELECT * FROM personas;");
4  $personas = $sentencia->fetchAll(PDO::FETCH_OBJ);
5  ?>
6  <!--Recordemos que podemos intercambiar HTML y PHP como queramos-->
7  <!DOCTYPE html>
8  <html lang="es">
9  <head>
10 <meta charset="UTF-8">
11 <title>Tabla de ejemplo</title>
12 <style>
13 table, th, td {
14 border: 1px solid black;
15 }
16 </style>
17 </head>
18 <body>
19 <table>
20 <thead>
21 <tr>
22 <th>ID</th>
23 <th>Nombre</th>
24 <th>Apellidos</th>
25 <th>Género</th>
26 </tr>
27 </thead>
28 <tbody>
29 <!--
30 Atención aquí, sólo esto cambiará
31 Pd: no ignores las llaves de inicio y cierre {}
32 -->
33 <?php foreach($personas as $persona){ ?>
34 <tr>
35 <td><?php echo $persona->id ?></td>
36 <td><?php echo $persona->nombre ?></td>
37 <td><?php echo $persona->apellidos ?></td>
38 <td><?php echo $persona->sexo ?></td>
39 </tr>
40 <?php } ?>
41 </tbody>
42 </table>
43 </body>
44 </html>
```

Por cierto, no es un error dejar sin punto y coma el echo. Pero recordemos que si estamos en donde las etiquetas se cierran (?>) no es necesario poner punto y coma. Ojo: sólo en la última expresión, no en todas.

8.6.4 Agregando enlaces

Para eliminar o actualizar necesitamos el id de la persona. Si no, ¿qué persona eliminaremos o editaremos?.

Gracias a la combinación con HTML creamos un link que dirija a cierto archivo, pasándole el id. Y luego, leer ese id.

Por el momento sólo hay que crear los enlaces, no hace falta explicar el funcionamiento. Así que la versión 3 de la tabla queda así:

```
1  <?php
2  include_once "base_de_datos.php";
3  $sentencia = $base_de_datos->query("SELECT * FROM personas;");
4  $personas = $sentencia->fetchAll(PDO::FETCH_OBJ);
5  ?>
6  <!--Recordemos que podemos intercambiar HTML y PHP como queramos-->
7  <!DOCTYPE html>
8  <html lang="es">
9  <head>
10 <meta charset="UTF-8">
11 <title>Tabla de ejemplo</title>
12 <style>
13 table, th, td {
14 border: 1px solid black;
15 }
16 </style>
17 </head>
18 <body>
19 <table>
20 <thead>
21 <tr>
22 <th>ID</th>
23 <th>Nombre</th>
24 <th>Apellidos</th>
25 <th>Género</th>
26 <th>Editar</th>
27 <th>Eliminar</th>
28 </tr>
29 </thead>
30 <tbody>
31 <!--
32 Atención aquí, sólo esto cambiará
33 Pd: no ignores las llaves de inicio y cierre {}
34 -->
35 <?php foreach($personas as $persona){ ?>
36 <tr>
37 <td><?php echo $persona->id ?></td>
38 <td><?php echo $persona->nombre ?></td>
39 <td><?php echo $persona->apellidos ?></td>
40 <td><?php echo $persona->sexo ?></td>
41 <td><a href="<?php echo "editar.php?id=" . $persona->id?>">Editar</a>
42 <td><a href="<?php echo "eliminar.php?id=" . $persona->id?>">Eliminar
   </a></td>
```



```

43 </tr>
44 <?php } ?>
45 </tbody>
46 </table>
47 </body>
48 </html>

```

Si acercas el cursor en donde dice `.eliminar` verás que te va a dirigir a un enlace como `.eliminar.php?id=1`. Pero ese id va cambiando en cada fila. Si le das click te dirá que no se ha encontrado el archivo, pues tenemos que crearlos a ambos: `editar.php` y `eliminar.php`.

8.6.5 Actualizando o editando datos

De todas las operaciones, en primer lugar debemos saber el id; luego, recuperar los datos para mostrarle al usuario lo que existe. Finalmente realizar la consulta UPDATE en la base de datos. Pero bueno, este paso es necesario. Así que vamos allá.

¿recuerdas que mandamos un id dentro del link en la tabla? ahora vamos a recuperarlo. Creamos el archivo `editar.php` y escribimos esto dentro de él:

```

1  <?php
2  if(!isset($_GET["id"])) exit();
3  $id = $_GET["id"];
4  include_once "base_de_datos.php";
5  $sentencia = $base_de_datos->prepare("SELECT * FROM personas WHERE id
   = ?;");
6  $sentencia->execute([$id]);
7  $persona = $sentencia->fetch(PDO::FETCH_OBJ);
8  if($persona === FALSE){
9  #No existe
10 echo "No existe alguna persona con ese ID!";
11 exit();
12 }
13 #Si la persona existe, se ejecuta esta parte del código
14 ?>
15 <!DOCTYPE html>
16 <html lang="es">
17 <head>
18 <meta charset="UTF-8">
19 <title>Registrar persona</title>
20 </head>
21 <body>
22 <form method="post" action="guardarDatosEditados.php">
23 <!-- Ocultamos el ID para que el usuario no pueda cambiarlo (en teorí
   a) -->
24 <input type="hidden" name="id" value="<?php echo $persona->id; ?>">
25
26 <label for="nombre">Nombre:</label>
27 <br>
28 <input value="<?php echo $persona->nombre ?>" name="nombre" required
   type="text" id="nombre" placeholder="Escribe tu nombre...">
29 <br><br>
30 <label for="apellidos">Apellidos:</label>

```

```

31 <br>
32 <input value="<?php echo $persona->apellidos ?>" name="apellidos"
    required type="text" id="apellidos" placeholder="Escribe tus
    apellidos...">
33 <br><br>
34 <label for="sexo">Género</label>
35 <select name="sexo" required name="sexo" id="sexo">
36 <!--
37 Para seleccionar una opción con defecto, se debe poner el atributo
    selected.
38 Usamos el operador ternario para que, si es esa opción, marquemos la
    opción seleccionada
39 -->
40 <option value="">--Selecciona--</option>
41 <option <?php echo $persona->sexo === 'M' ? "selected='selected'": ""
    ?> value="M">Masculino</option>
42 <option <?php echo $persona->sexo === 'F' ? "selected='selected'": ""
    ?> value="F">Femenino</option>
43 </select>
44 <br><br><input type="submit" value="Guardar cambios">
45 </form>
46 </body>
47 </html>
48
49 <?php
50 if(!isset($_GET["id"])) exit();
51 $id = $_GET["id"];
52 include_once "base_de_datos.php";
53 $sentencia = $base_de_datos->prepare("SELECT * FROM personas WHERE id
    = ?;");
54 $sentencia->execute([$id]);
55 $persona = $sentencia->fetch(PDO::FETCH_OBJ);
56 if($persona === FALSE){
57 echo "No existe persona con ese ID";
58 exit();
59 }
60 // Si la persona existe, se ejecuta esta parte del código
61 ?>
62 <!DOCTYPE html>
63 <html lang="es">
64 <head>
65 <meta charset="UTF-8">
66 <title>Registrar persona</title>
67 </head>
68 <body>
69 <form method="post" action="guardarDatosEditados.php">
70 <!-- Ocultamos el ID para que el usuario no pueda cambiarlo (en teorí
    a) -->
71 <input type="hidden" name="id" value="<?php echo $persona->id; ?>">
72
73 <label for="nombre">Nombre:</label>
74 <br>

```

```

75 <input value="<?php echo $persona->nombre ?>" name="nombre" required
    type="text" id="nombre" placeholder="Escribe tu nombre...">
76 <br><br>
77 <label for="apellidos">Apellidos:</label>
78 <br>
79 <input value="<?php echo $persona->apellidos ?>" name="apellidos"
    required type="text" id="apellidos" placeholder="Escribe tus
    apellidos...">
80 <br><br>
81 <label for="sexo">Género</label>
82 <select name="sexo" required name="sexo" id="sexo">
83 <!--
84 Para seleccionar una opción con defecto, se debe poner el atributo
    selected.
85 Usamos el operador ternario para que, si es esa opción, marquemos la
    opción seleccionada
86 -->
87 <option value="">--Selecciona--</option>
88 <option <?php echo $persona->sexo === 'M' ? "selected='selected'": ""
    ?> value="M">Masculino</option>
89 <option <?php echo $persona->sexo === 'F' ? "selected='selected'": ""
    ?> value="F">Femenino</option>
90 </select>
91 <br><br><input type="submit" value="Guardar cambios">
92 </form>
93 </body>
94 </html>

```

Si te fijas bien, ahora estamos recibiendo el id por medio del arreglo superglobal `$_GET`. Y accedemos a él a través de su posición como cadena, así:

```
$_GET["id"]
```

¡Justo como leíamos a `$_POST`! Luego, hacemos una búsqueda en donde el id sea igual al proporcionado. Si no existe nada con ese id (porque un usuario puede cambiar ese id a 4654654, cosa que en teoría no existe) entonces lo advertimos.

Veamos ahora el formulario, HTML provee una forma de dar valores por defecto a los campos utilizando el atributo "value." así que llenamos estos valores con los datos que obtuvimos, en caso de que existan.

También agregamos un input de tipo oculto, ahí guardaremos el ID para que el usuario, en teoría, no pueda cambiarlo.

Y finalmente observa el action, ahora no irá a `nuevaPersona.php` sino a `guardarDatosEditados.php`. Éste último archivo vamos a programarlo ahora mismo.

8.6.6 Recibiendo datos para editar

Este será casi igual al de `nuevaPersona.php` salvo que ahora también necesitamos el id, y será un UPDATE en lugar de un INSERT. Veamos el código y luego lo explicamos:

```

1 <?php
2 #Salir si alguno de los datos no está presente
3 if(
4 !isset($_POST["nombre"]) ||

```

```

5  !isset($_POST["apellidos"]) ||
6  !isset($_POST["sexo"]) ||
7  !isset($_POST["id"])
8  ) exit();
9  #Si todo va bien, se ejecuta esta parte del código...
10 include_once "base_de_datos.php";
11 $id = $_POST["id"];
12 $nombre = $_POST["nombre"];
13 $apellidos = $_POST["apellidos"];
14 $sexo = $_POST["sexo"];
15 $sentencia = $base_de_datos->prepare("UPDATE personas SET nombre = ?,
    apellidos = ?, sexo = ? WHERE id = ?;");
16 $resultado = $sentencia->execute([$nombre, $apellidos, $sexo, $id]);
    # Pasar en el mismo orden de los ?
17 if($resultado === TRUE) echo "Cambios guardados";
18 else echo "Algo salió mal. Por favor verifica que la tabla exista, as
    í como el ID del usuario";
19 ?>
20
21 <?php
22 #Salir si alguno de los datos no está presente
23 if(
24 !isset($_POST["nombre"]) ||
25 !isset($_POST["apellidos"]) ||
26 !isset($_POST["sexo"]) ||
27 !isset($_POST["id"])
28 ) exit();
29 #Si todo va bien, se ejecuta esta parte del código...
30 include_once "base_de_datos.php";
31 $id = $_POST["id"];
32 $nombre = $_POST["nombre"];
33 $apellidos = $_POST["apellidos"];
34 $sexo = $_POST["sexo"];
35 $sentencia = $base_de_datos->prepare("UPDATE personas SET nombre = ?,
    apellidos = ?, sexo = ? WHERE id = ?;");
36 $resultado = $sentencia->execute([$nombre, $apellidos, $sexo, $id]);
    # Pasar en el mismo orden de los ?
37 if($resultado === TRUE) echo "Cambios guardados";
38 else echo "Algo salió mal. Por favor verifica que la tabla exista, as
    í como el ID del usuario";
39 ?>

```

Usamos el ID para hacer un Where dentro de la consulta. Y actualizamos todos los valores. De todos modos, no hay problema, pues si el usuario no toca el valor se queda tal y como estaba.

8.6.7 Eliminando datos

Pequeña gran nota: repito que esto es un post más que nada educativo. Nunca pero nunca debemos hacer que se elimine usando un link, y mucho menos sin confirmación. ¿te imaginas que Facebook tuviera esa opción para, por ejemplo, eliminar mensajes con alguna persona? supongamos que el link fuera algo como `eliminar_mensajes.php?idPersona=111`

Ahora yo te digo que visites ese link, pero antes lo pongo en un acortador para que no haya

sospechas. Una vez que hagas click en él, adiós a esa conversación. Obviamente no perdemos nada con eliminar una conversación (al menos que incrimine a alguien, tenga valor sentimental, etcétera) pero basta con esto para darnos una idea.

En este ejemplo eliminamos con el valor pasado por un link y no hay vuelta atrás, pero igual no perdemos nada. Para casos reales, es mejor hacerlo usando POST, pues ahí los datos no se ven en la URL. Además, podemos mandar una confirmación, comprobar si el usuario tiene permisos, etcétera.

Pero eso es otra historia, veamos ahora sí el último pilar para armar este crud. Por cierto, podríamos poner un confirm con javascript, o algo así, pero sería mezclar código y confundir al lector.

```
1  <?php
2  if(!isset($_GET["id"])) exit();
3  $id = $_GET["id"];
4  include_once "base_de_datos.php";
5  $sentencia = $base_de_datos->prepare("DELETE FROM personas WHERE id =
   ?;");
6  $resultado = $sentencia->execute([$id]);
7  if($resultado === TRUE) echo "Eliminado correctamente";
8  else echo "Algo salió mal";
9  ?>
10
11 <?php
12 if(!isset($_GET["id"])) exit();
13 $id = $_GET["id"];
14 include_once "base_de_datos.php";
15 $sentencia = $base_de_datos->prepare("DELETE FROM personas WHERE id =
   ?;");
16 $resultado = $sentencia->execute([$id]);
17 if($resultado === TRUE) echo "Eliminado correctamente";
18 else echo "Algo salió mal";
19 ?>
```

La operación que más me gusta es eliminar, pues sólo necesitamos el ID. Espero que el código se explique por sí mismo, porque si no, ni yo mismo podría hacerlo.

8.6.8 Probando y descargando

Sería un pecado explicar todo esto y no dejar los archivos para que los pruebes por ti mismo. Aquí dejo el proyecto completo, sólo pégalo y extráelo en la carpeta pública de tu servidor Apache (htdocs o public_html).

mysql_pdo_php_crud

También puedes probarlo en línea.

CONCLUSIÓN. Podemos agregar unos links de navegación para ir, por ejemplo, de editar a listar todos, etcétera. Pero eso lo dejo para que el usuario lo haga. También agrega un estilo CSS. O incluso crear un servicio web con PHP y consumirlo desde cualquier lugar, o hacerlo con Javascript utilizando AJAX.

Finalmente, si quieres un ejemplo un poco más avanzado, te invito a leer Pequeño, muy pequeño sistema de ventas con PHP.

8.6.9 Consultas

PDO proporciona dos métodos para crear consultas:

8.6.9.1 Consultas no preparadas

Con el método `query()` tenemos una alternativa al uso de consultas preparadas. La utilizamos mayormente cuando la consulta no tiene parámetros externos. Ya que este método no escapa automáticamente los parámetros. Además, no nos beneficiamos de la mejora de rendimiento de consultas múltiples.

Este método ejecuta la consulta (no necesita preparación) y devuelve un objeto `PDOStatement` en caso de acierto y `False` en caso de error. Como vemos posteriormente, las consultas preparadas necesitan 2 pasos antes de recoger el resultado: preparar (`prepare()`) y ejecutar (`execute()`).

```
1 try {
2     $con = new PDO('mysql:dbname=demo';host=localhost', $user, $password)
3     ;
4     $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5     $stmt = $con->query('SELECT nombre FROM usuarios');
6 } catch(PDOException $e) {
7     echo 'Error: ' . $e->getMessage();
8 }
```

Este método no es aconsejable para sentencias con parámetros externos. Pero si quiere utilizarlo, debemos acompañarlo del método `quote()`. Para indicar que queremos escapar (anteponer 'á caracteres problemáticos). Es similar a utilizar `mysqli_real_escape_string()` o `real_escape_string()` si utiliza la interfaz orientada a objetos.

```
1 try {
2     $con = new PDO('mysql:dbname=demo';host=localhost', $user,
3     $password);
4     $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5     $stmt = $con->query('SELECT nombre FROM usuario WHERE nombre like '
6     . $con->quote($cad));
7 } catch(PDOException $e) {
8     echo 'Error: ' . $e->getMessage();
9 }
```

8.6.9.2 Consultas preparadas

El método necesario para preparar consultas preparadas es `prepare()`. Se encarga automáticamente de escapar los parámetros que se pasen, evitándonos problemas con la inyección de SQL. Esta función devuelve un `PDOStatement` sobre el cual ejecutaremos la función `fetch()` para mostrar su valor. Pero antes vamos a ver como pasar parámetros. Importante es saber que `prepare()` va a devolver un objeto `PDOStatement` en caso de que no haya error al preparar la consulta (`False` en caso de error). Y este objeto es el que vamos a utilizar para realizar las posteriores vinculaciones, ejecuciones y recogida de resultados.

Los parámetros de entrada de datos irán precedidos por (:) y posteriormente el nombre del parámetro de sustitución.

```
$stmt = $con->prepare('SELECT * FROM tabla WHERE Nombre=:campo_nombre and Apellido=:campo_apellido');
```

Otra alternativa es usar (?) para los parámetros de entrada:

```
$stmt = $con->prepare('SELECT * FROM tabla WHERE Nombre=? and Apellido=?');
```

Utilicemos la alternativa que queramos, el siguiente paso es vincular un valor a los parámetros de sustitución. Y para ello utilizaremos la función bindValue() o bindParam().

```
1 public bool PDOStatement::bindValue ( mixed $parameter , mixed $value
  [, int $data_type = PDO::PARAM_STR ] )
2 public bool PDOStatement::bindParam ( mixed $parameter , mixed &
  $variable [, int $data_type = PDO::PARAM_STR [, int $length [,
  mixed $driver_options ]]] )
```

donde \$data_type (parámetro opcional) puede ser:

```
1 - PDO::PARAM_BOOL
2 - PDO::PARAM_NULL
3 - PDO::PARAM_INT
4 - PDO::PARAM_STR (por defecto)
5 - PDO::PARAM_LOB
```

Por lo tanto si hemos utilizado la sintaxis (:)

```
1 $stmt->bindValue(':campo_nombre', $nombre);
2 $stmt->bindValue(':campo_apellido', $apellido);
```

O si hemos optado por la sintaxis (?).

```
1 $stmt->bindValue(1, $nombre);
2 $stmt->bindValue(2, $apellido);
```

Como observaremos el primer parámetro (\$parameter) en este caso se indica con la posición en la consulta del campo a vincular. A diferencia de si usamos la sintaxis (:) donde si especificamos el campo a vincular en la consulta.

Las funciones bindParam() y bindValue() son muy similares, salvo por:

1. Con bindParam() sólo pasamos variables y no valores directamente. Funciona con las variables ya que permite que parámetros por referencia"(y un valor que no es una referencia"válida en PHP).
2. Con bindValue pasamos tanto valores como variables.

8.6.9.3 Ejecución

Solo si hemos utilizado una consulta preparada, necesitamos ejecutar nuestra consulta con la siguiente función antes de poder recuperar los resultados. Si la consulta se realiza con query(), y por lo tanto no es preparada, podremos directamente recoger los datos con una de las funciones del apartado 6.

```
public bool PDOStatement::execute ([ array $input_parameters ] )
```

Una característica importante de esta función es que permite opcionalmente introducir los parámetros a vincular. Por lo tanto si no queremos hacer uso de las funciones de vinculación `bindParam()` o `bindValue()`, podemos pasar los parámetros directamente a esta función.

```
1 | $stmt = $gbd->prepare('SELECT name, colour, calories FROM fruit
2 | WHERE calories < :calories AND colour = :colour');
3 | $stmt->execute(array(':calories' => $calorías, ':colour' => $color));
```

Esta función devuelve un objeto `True` o `False` en caso de éxito o de error a realizar la consulta.

8.6.9.4 Recolección de datos

Como hemos comentado cuando presentamos `prepare()` o `query()`, tenemos un objeto de tipo `PDOStatement` (`$stmt`), que hemos utilizando para las vinculaciones y ejecuciones en el caso de consultas preparadas. Y para recuperar el resultado de la ejecución de una consulta también haremos uso de él.

En `MySQLi` tenemos 3 funciones (tanto para interfaz orientada a objetos como para la procedimental) para obtener los resultado de un objeto `mysqli_result`:

```
1 | - array mysqli_result::fetch_assoc ( void ) / array
   |   mysqli_fetch_assoc ( mysqli_result $result )
2 | - mixed mysqli_result::fetch_array ( [ int $tiporesultado =
   |   MYSQLI_BOTH ] ) /
3 | mixed mysqli_fetch_array ( mysqli_result $result [, int
   |   $tiporesultado = MYSQLI_BOTH ] )
4 | - object mysqli_result::fetch_object ( [ string $nombre_clase [, array
   |   $params ]] ) /
5 | object mysqli_fetch_object ( mysqli_result $result [, string
   |   $class_name [, array $params ]] )
```

Y con `PDOStatement` tenemos 3 métodos principales para acceder al resultado de una ejecución. Aunque los siguientes métodos tienen más parámetros presentamos los principales.

8.6.9.5 Obtener la siguiente fila de un conjunto de resultados

Por lo que tendremos que utilizar un bucle para recorrer todos los resultados (filas) de la ejecución de la consulta.

```
public mixed PDOStatement::fetch ( [ int $fetch_style ] )
```

Dependiendo del valor del parámetro que especifiquemos, el resultado es de un tipo u otro. Aunque habitualmente sólo utilizaremos uno de los 3 siguientes valores, que devuelven el resultado en forma de array:

1. `PDO::FETCH_ASSOC`: devuelve un array indexado por los nombres de las columnas del conjunto de resultados.
2. `PDO::FETCH_NUM`: devuelve un array indexado por el número de columna tal como fue devuelto en el conjunto de resultados, comenzando por la columna 0.
3. `PDO::FETCH_BOTH` (predeterminado): devuelve un array indexado tanto por nombre de columna, como numéricamente con índice de base 0 tal como fue devuelto en el conjunto de resultados.

Ejemplo

```
1  //...
2  $stmt = $con->prepare("SELECT id, nombre, email FROM usuario");
3  $stmt->execute();
4  /* Prueba de tipos de PDOStatement::fetch */
5  echo "PDO::FETCH_ASSOC: ";
6  echo "Devolver la siguiente fila como un array indexado por nombre de
   columna\n";
7  $result = $stmt->fetch(PDO::FETCH_ASSOC);
8  print_r($result);
9  echo "\n";
10 echo "PDO::FETCH_NUM: ";
11 echo "Devolver la siguiente fila como un array indexado por número de
   columna\n";
12 $result = $stmt->fetch(PDO::FETCH_BOTH);
13 print_r($result);
14 echo "\n";
15 echo "PDO::FETCH_BOTH: ";
16 echo "Devolver la siguiente fila como un array indexado por nombre y
   número de columna\n";
17 $result = $stmt->fetch(PDO::FETCH_BOTH);
18 print_r($result);
19 echo "\n";
```

Salida

```
1  PDO::FETCH_ASSOC: Devuelve la siguiente fila como un array indexado
   por nombre de columna
2  Array(
3  [id] => 1
4  [nombre] => jose
5  [email] => prueba@ejemplo.com
6  )
7  PDO::FETCH_NUM: Devolver la siguiente fila como un array indexado por
   número de columna";
8  Array(
9  [0] => 1
10 [1] => jose
11 [2] => prueba@ejemplo.com
12 )
13 PDO::FETCH_BOTH: Devolver la siguiente fila como un array indexado
   por nombre y número de columna
14 Array(
15 [id] => 1
16 [0] => 1
17 [nombre] => jose
18 [1] => jose
19 [email] => prueba@ejemplo.com
20 [2] => prueba@ejemplo.com
21 )
```

Ejemplo con query() y bucle para obtener todos los resultados.

```
1  $sql = "SELECT * FROM usuario";
```

```

2 $stmt= $conn->query($sql);
3 while($fila = $stmt->fetch(PDO::FETCH_ASSOC)) {
4 echo $fila['id']. ' - '. $fila['nombre']. ' - '. $fila['email']. ' -
   '. $fila['direccion']. '<br />';
5 }

```

8.6.9.6 Obtener un array con todas las filas del resultado

Mientras que con la anterior función obteníamos un array con los campos de una fila. Ahora una matriz (array de arrays) de dos dimensiones.

```
public array PDOStatement::fetchAll ([ int $fetch_style])
```

Al igual que con la función anterior fetch() dependiendo de el valor que pasemos como \$fetch_style el array tendrá una forma u otra. Nuevamente FETCH_BOTH es el valor por defecto.

```

1 $stmt = $con->prepare("SELECT id, nombre, emial FROM usuario");
2 $stmt->execute();
3 /* Usaremos el estilo por defecto en la función fetch_all */
4 print("Obtener todas las filas restantes del conjunto de resultados:\n");
5 $resultado = $stmt->fetchAll();
6 print_r($resultado);

```

Obtener todas las filas restantes del conjunto de resultados: Array([0] =>Array ([id] =>1 [0] =>1 [nombre] =>jose [1] =>jose [email] =>prueba@ejemplo.com [2] =>prueba@ejemplo.com) [1] =>Array([id] =>2 [0] =>2 [nombre] =>juan [1] =>juan [email] =>prueba2@ejemplo.com [2] =>prueba2@ejemplo.com))

8.6.9.7 Obtener la siguiente fila y devolverla como un objeto

Al igual que pasaba con fetch_object de la extensión mysqli, podemos (primer parámetro opcional) pasar los datos de la fila a un objeto y así llamar a métodos de la clase instanciada para trabajar con ellos. Además podríamos (segundo parámetro opcional) pasar datos al constructor de la clase.

Los campos de la fila del conjunto resultado se agregan automáticamente como atributos públicos de la clase. Por lo tanto podremos acceder a ellos dentro de la clase, sin tenerlos que definir. Pero es una buena practica definirlos en la clase. Y si se definen, tienen que coincidir con el nombre de la columna.

```
public mixed PDOStatement::fetchObject ([ string $class_name = "stdClass" [, array $ctor_args ] ] )
```

Si no instanciamos ninguna clase simplemente accederemos a las columna de la fila como si esta fuera un objeto y las columnas propiedades.

```

1 class animal{
2     public $id;
3     public $tipo;
4     public $nombre;
5     public function capitalizarTipo() {
6         return ucwords($this->tipo);
7     }
8 }

```

```

9 | try {
10 |     $dbh = new PDO("mysql:host=$hostname;dbname=zoo", $username,
    |         $password);
11 |     $sql = "SELECT * FROM animales";
12 |     $stmt = $dbh->query($sql);
13 |     while ($animal = $stmt->fetchObject('animal')) {
14 |         echo $animal->id . '<br />';
15 |         echo $animal->capitalizarTipo() . '<br />';
16 |         echo $animal->nombre;
17 |     }
18 | } catch (PDOException $e) {
19 |     echo $e->getMessage();
20 | }

```

8.6.10 Otras funciones útiles de PDO

Existen tres funciones para obtener información de la consulta ejecutada sin recorrer los resultados:

8.6.10.1 Obtener el id de la última fila o secuencia insertada en la base de datos

```
public string PDO::lastInsertId ([ string $name = NULL ] )
```

El parámetro \$name opcional, indica el nombre de columna. Como PDO depende de los drivers para cada gestor de base de datos soportado, en alguno, quizás sea necesario especificar el nombre de columna que almacena los identificadores. Como ocurre en PostgreSQL.

```

1 | $conn = new PDO('mysql:dbname=test;host=127.0.0.1', 'user', 'password
    | ');
2 | $stmt = $conn->prepare('INSERT INTO test (name) VALUES (:name)');
3 | $stmt->execute([':name' => 'foo']);
4 | var_dump($conn->lastInsertId());

```

Recuerda que la función es de la clase PDO y no de PDOStatement.

8.6.10.2 Número de filas afectadas por la última sentencia de inserción, borrado o actualización

Función de la clase PDOStatement.

```

1 | public int PDOStatement::rowCount ( void )
2 | $stmt = $con->prepare('DELETE FROM usuario');
3 | $stmt->execute();
4 | $count = $stmt->rowCount();
5 | print("se borraron: $count filas.\n");

```

Esta función es útil para borrar una fila, por ejemplo un único usuario. De esta manera comprobamos cuantas filas se han borrado; si el resultado es cero ha fallado y si es 1 ha funcionado.

PDO depende de los drivers de las bases de datos que soporta, pero no asegura que para todas las bases de datos la función rowCount() funcione correctamente para sentencias de tipo SELECT. Sin embargo, tampoco es necesaria para consultas de selección, ya que utilizamos otras soluciones:

1. Para conocer el número total de filas es mejor averiguarlo usando la base de datos. Después de ejecutar la consulta utilizamos la función `fetchColumn([$num_columna = 0])`, que

devuelve el resultado de la columna que se le indique como parámetro opcional (cero por defecto) en la primera fila de resultados.

```
1 | $sql = "SELECT COUNT(*) FROM fruit WHERE calories > 100";
2 | $res = $conn->query($sql){
3 |   if ($res->fetchColumn() > 0) {
4 |   }
```

2. Si lo que queremos es utilizar un bucle para obtener cada fila y trabajar con ellas, el bucle termina cuando no haya más filas. Si desde un principio no hay filas, el bucle no ejecuta nada.
3. Si la comprobación va destinada a comprobar si existe un usuario. Simplemente obtenemos la única fila a obtener con `fetch()`. Si devuelve false no se ha podido obtener alguna fila; así conocemos si existe o no el usuario. Por lo que tampoco necesitamos usar `rowCount()`.

```
1 | public function getUsuario($usuario, $password){
2 |   $datos = $this->_db->query(
3 |     "select * from usuarios " .
4 |     "where usuario = '$usuario' " .
5 |     "and pass = ' ' . Hash::getHash('sha1',$password,HASH_KEY) ." ' "
6 |   );
7 |   //fetch devuelve false en caso de consulta sin resultados
8 |   return $datos->fetch();
9 | }
10 | ...
11 | $row = $this->getUsuario($usuario, $pass);
12 | if (!$row) {
13 |   ...
14 | }
```

Estas funciones tienen su equivalente en MySQLi en consultas normales o preparadas.

Ejemplos

Inserción

```
1 | try {
2 |   $dbh = new PDO("mysql:host=$hostname;dbname=$dbname", $username,
3 |     $password);
4 |   $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5 |   $stmt = $dbh->prepare("INSERT INTO table(field1,field2,field3) VALUES
6 |     (:field1,:field2,:field3)");
7 |   $stmt->execute(array(':field1' => $field1, ':field2' => $field2, ':
8 |     field3' => $field3));
9 |   $affected_rows = $stmt->rowCount();
10 | }
11 | catch(PDOException $e) {
12 |   echo $sql . ' <br />' . $e->getMessage();
13 | }
```

Inserción múltiple

```
1 | try {
2 |   $dbh = new PDO("mysql:host=$hostname;dbname=$dbname", $username,
3 |     $password);
```

```

3 | $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
4 | $values = array('bob', 'alice', 'lisa', 'john');
5 | $name = '';
6 | $stmt = $dbh->prepare("INSERT INTO table(`name`) VALUES(:name)");
7 | $stmt->bindParam(':name', $name, PDO::PARAM_STR);
8 | foreach($values as $name) {
9 |     $stmt->execute();
10 | }
11 | }
12 | catch(PDOException $e) {
13 |     echo $sql . "<br />" . $e->getMessage();
14 | }

```

Borrado

```

1 | try {
2 |     $dbh = new PDO("mysql:host=$hostname;dbname=$dbname", $username,
3 |         $password);
4 |     $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5 |     $stmt = $dbh->prepare("DELETE FROM table WHERE id=:id");
6 |     $stmt->bindValue(':id', $id, PDO::PARAM_INT);
7 |     $stmt->execute();
8 |     $affected_rows = $stmt->rowCount();
9 | }
10 | catch(PDOException $e) {
11 |     echo $sql . "<br />" . $e->getMessage();
12 | }

```

Actualización

```

1 | try {
2 |     $dbh = new PDO("mysql:host=$hostname;dbname=$dbname", $username,
3 |         $password);
4 |     $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5 |     $stmt = $dbh->prepare("UPDATE table SET name=? WHERE id=?");
6 |     $stmt->execute(array($name, $id));
7 |     $affected_rows = $stmt->rowCount();
8 | }
9 | catch(PDOException $e) {
10 |     echo $sql . "<br />" . $e->getMessage();
11 | }

```

Recuerda que si en las funciones `execute()`, `bindValue()` o `bindParam` no especificamos el tipo del dato, éste se trata por defecto como `PARAM_STR`. Por lo tanto se filtran como cadenas.

8.6.11 Transacciones

Ya explicamos las transacciones con la extensión `MySqli`; con `PDO` funcionan de una manera muy similar. En resumen, son un mecanismo que asegura que todas y cada de las consultas van a ser ejecutadas, de forma segura y sin interferencia de otras conexiones.

En el caso de que falle alguna consulta, la transacción no se llevará a cabo.

Pero recuerda que `PDO` trabaja como capa de abstracción sobre múltiples tipos de gestores de bases de datos así que cada uno de estos gestores tienen sus reglas y es posible que alguno no admita el mecanismo de transacciones. Con `MySQL`, por ejemplo, hay que tener cuidado con el

tipo de motores de almacenamiento de las tablas. Ya que todos no soportan las transacciones. Los tipos de tablas MyISAM no aceptan transacciones. Éstas, sólo se ejecutan sobre tablas InnoDB. Veamos el proceso de crear transacciones:

8.6.11.1 Iniciar la transacciones

PDO, al igual que con MySQLi, ejecuta por defecto el modo "autocommit". Lo que significa que cada consulta ejecutada se trata como una transacción implícitamente. Por lo tanto el primer paso para usar las transacciones es desactivar este modo. Y así decidir donde empieza y acaba la transacción. Ésta no se ejecuta hasta que se le indique.

Para ello utilizaremos la siguiente función:

```
public bool PDO::beginTransaction ( void )
```

Que devuelve True en caso de acierto y False en caso de error. Además, Si el controlador subyacente no admite transacciones, se lanza una PDOException (independientemente de la configuración del manejo de errores: es una condición de error serio).

8.6.11.2 Crear consultas y finalizar o revertir transacción

Una vez creadas las consultas, finalizamos la transacción si no ha habido ningún error. O la revertimos, en caso de algún error.

Antes de mostrar un ejemplo completo vamos a presentar la función de la clase PDO: `public int PDO::exec (string $statement)`

Dicha función ejecuta una sentencia sql y devuelve el numero de filas afectadas por dicha ejecución. Como una alternativa similar a `query()`. Sin embargo, no devuelve los resultados de una sentencia SELECT. Su funcionalidad se limita a consultas donde no obtengamos un conjunto de resultados. Es ideal para ejecutar transacciones, donde lo normal no ejecutar consultas SELECT.

Función para ejecutar la transacción. Hasta que no se indique ninguna de las consultas de la transacción a ejecutar.

```
public bool PDO::commit ( void )
```

Función para revertir la transacción.

```
public bool PDO::rollBack ( void )
```

Hay que tener en cuenta que si confirma una transacción o la revierte, devolvemos la conexión de base de datos a modalidad "autocommit" hasta que la siguiente llamada a `PDO::beginTransaction()` inicia una nueva transacción.

```
1 try {
2     $dbh = new PDO("mysql:host=$hostname;dbname=$dbname", $username,
3         $password);
4     echo 'Conectado a la base de datos<br />';
5     /** Poner el modo de errores para detectar excepciones **/
6     $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
7     /** Empezamos la transacción **/
8     $dbh->beginTransaction();
9     /** Creamos tabla **/
10    $table = "CREATE TABLE animals ( animal_id MEDIUMINT(8) NOT NULL
        AUTO_INCREMENT PRIMARY KEY,
        animal_type VARCHAR(25) NOT NULL,
```

```

11     animal_name VARCHAR(25) NOT NULL
12 );
13 $dbh->exec($table);
14 /** insertamos datos */
15 $dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES
16     ('emu', 'johnny')");
17 $dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES
18     ('funnel web', 'bruce')");
19 $dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES
20     ('lizard', 'liz')");
21 $dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES
22     ('dingo', 'dango')");
23 $dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES
24     ('kangaroo', 'bob')");
25 $dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES
26     ('wallaby', 'wally')");
27 $dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES
28     ('wombat', 'rick')");
29 $dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES
30     ('koala', 'mowgli')");
31 $dbh->exec("INSERT INTO animals (animal_type, animal_name) VALUES
32     ('kiwi', 'tonny')");
33
34 /** Enviamos la transacción */
35 $dbh->commit();
36 echo 'Datos insertados correctamente<br />';
37 }
38 catch(PDOException $e) {
39     /** revertimos transacción por algun fallo */
40     $dbh->rollback();
41     echo $sql . '<br />' . $e->getMessage();
42 }

```

Capítulo 9

Bases de datos con PDO

9.1 Introducción

Usamos PDO para una conexión flexible y segura en PHP.

PDO significa PHP Data Objects, Objetos de Datos de PHP, una extensión para acceder a bases de datos. PDO permite acceder a diferentes sistemas de bases de datos con un controlador específico (MySQL, SQLite, Oracle...) mediante el cual se conecta. Independientemente del sistema utilizado, se emplearán siempre los mismos métodos, lo que hace que cambiar de uno a otro resulte más sencillo.

Para ver los controladores (drivers) disponibles en tu servidor, puedes emplear el método `getAvailableDrivers()`: `print_r(PDO::getAvailableDrivers());`

El sistema PDO se fundamenta en 3 clases: PDO, PDOStatement y PDOException. La clase PDO se encarga de mantener la conexión a la base de datos y otro tipo de conexiones específicas como transacciones, además de crear instancias de la clase PDOStatement. Es ésta clase, PDOStatement, la que maneja las sentencias SQL y devuelve los resultados. La clase PDOException se utiliza para manejar los errores.

La extensión PDO facilita trabajar con bases de datos, ya que abstrae el tipo de gestor de base de datos a utilizar. Esto quiere decir que con el uso de PDO, si decides migrar tu aplicación de MySQL a PostgreSQL basta con cambiar únicamente el tipo de base de datos a conectar, y el resto de la aplicación sigue funcionando igual. Esta abstracción del acceso de datos convierte a esta extensión en un elemento muy importante para la portabilidad de la aplicación/web.

PDO además utiliza consultas preparadas. Y como vimos en el caso de la extensión MySQLi proporciona una herramienta muy importante en seguridad y rendimiento.

1. Se encargan de 'desinfectar' los datos automáticamente.
2. La separación de lógica y datos hace que la consulta sea analizada/preparada una única vez y se pueda ejecutar múltiples veces con los mismos datos o similares. Evitando cada vez el ciclo de análisis, compilación y optimización que ocurriría en cada consulta no preparada.
3. Al igual que MySQLi, proporciona el uso de transacciones.

En lo personal recomiendo PDO, pues es orientado a objetos.

Para interactuar con la base de datos necesitamos conectarnos a la misma. Para ello, usamos un

archivo y después simplemente lo incluimos en donde queramos usarlo. Voy a poner el código aquí, y lo explico más abajo.

```
1 <?php
2 $contraseña = "";
3 $usuario = "root";
4 $nombre_base_de_datos = "pruebas";
5 try {
6     $base_de_datos = new PDO('mysql:host=localhost;dbname=' .
7         $nombre_base_de_datos, $usuario, $contraseña);
8 } catch(Exception $e) {
9     echo "Ocurrió algo con la base de datos: " . $e->getMessage();
10 }
```

o usamos:

```
1 <?php
2 $contraseña = "";
3 $usuario = "root";
4 $nombre_base_de_datos = "pruebas";
5 try {
6     $base_de_datos = new PDO('mysql:host=localhost;dbname=' .
7         $nombre_base_de_datos, $usuario, $contraseña);
8 } catch(Exception $e) {
9     echo "Ocurrió algo con la base de datos: " . $e->getMessage();
10 }
```

Como vemos, el código es simple. Creamos un objeto PDO y pasamos 3 parámetros:

1. Nombre del origen de datos
2. El usuario
3. La contraseña

PDO cambia de gestor de base de datos en una sola línea, sólo cambiamos el primer parámetro. Lo ponemos en un try/catch porque en algunas ocasiones puede que escribimos mal el usuario, contraseña, base de datos, etcétera.

9.1.1 Especificar el tratamiento de errores

Por defecto PDO viene configurado para que no se muestre ningún error. Por lo que tras cada consulta (incluida la conexión) tendríamos que hacer uso de los métodos `errorCode()` y `errorInfo()`.

`//errorInfo` devuelve un array con información del error `print_r($con->errorInfo());`

Como ayuda, PDO activa el uso de excepciones. De tal manera que si hay algún error en conexión o consulta, activa una excepción que capturamos. Para ello, utilizamos el método `setAttribute` y el atributo `PDO::ATTR_ERRMODE`. Acompañando a dicho atributo indicamos uno de los 3 siguientes posibles valores:

1. PDO::ERRMODE_SILENT es el valor por defecto. Con dicho valor no se lanzará ningún error ni excepción. Y como se ha comentado, el programador es el encargado de comprobar mediante errorCode() y errorInfo() los posibles errores.
2. PDO::ERRMODE_EXCEPTION es el valor que habilitará que se lancen excepciones. Es el valor que nos interesa.
3. PDO::ERRMODE_WARNING genera un error E_WARNING de PHP si ocurre algún error.

```
$con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

9.1.2 Conectar a una base de datos con PDO

El primer argumento de la clase PDO es el DSN, Data Source Name, en el cual se han de especificar el tipo de base de datos (mysql), el host (localhost) y el nombre de la base de datos (se puede especificar también el puerto).

Para cada base de datos existe un manejador (driver) específico, que debe estar habilitado en el archivo de configuración de PHP (el archivo php.ini). Los manejadores se administran mediante extensiones de PHP, las cuales tienen nombres finalizando con dll en Windows y con so en Unix. Todas estas extensiones deben existir en el directorio de extensiones de PHP. Generalmente las extensiones php_pdo y php_pdo_sqlite estarán habilitadas por omisión.

Diferentes sistemas de bases de datos tienen distintos métodos para conectarse. La mayoría se conectan de forma parecida a como se conecta a MySQL:

```
1 | try {
2 |     $dsn = "mysql:host=localhost;dbname=$dbname";
3 |     $dbh = new PDO($dsn, $user, $password);
4 | } catch (PDOException $e){
5 |     echo $e->getMessage();
6 | }
```

DBH significa Database Handle, y es el nombre de variable que se suele utilizar para el objeto PDO.

Para cerrar una conexión: `$dbh = null;`

Para realizar una nueva conexión se crea una instancia del objeto PDO. Este constructor acepta una serie de parámetros de conexión (string de conexión) que pueden ser específicos para cada sistema de bases de datos.

Si no se logra establecer la conexión se produce una excepción (PDOException). Si la conexión es exitosa, una instancia de PDO será devuelta. La conexión permanece activa por todo el ciclo de vida del objeto PDO. Para cerrar la conexión, se debe destruir el objeto asegurándose que toda referencia sea eliminada, o bien, PHP cerrará la conexión automáticamente cuando el programa finalice.

Si se desea hacer una conexión persistente, que no sea eliminada al final de la ejecución del programa, es necesario habilitar la opción PDO::ATTR_PERSISTENT en el arreglo de las opciones de la conexión.

```
1 | <?php
2 | try {
3 |     $dbh = new PDO('sqlite:test.db');
```

```

4 | $dbh->exec("CREATE TABLE countries (name TEXT, area INTEGER,
   | population INTEGER, density REAL)");
5 | $dbh = null;
6 | } catch (PDOException $e) {
7 |     print "Error!: " . $e->getMessage() . "<br/>";
8 |     die();
9 | }
10 | ?>

```

Note que en el ejemplo anterior la base de datos podría ser creada usando el comando `sqlite3 test.db` (si está disponible en el ambiente). Además, el archivo `test.db` como el directorio en que se encuentra, deben tener derechos de escritura.

Puedes ver más información acerca de como conectarse a determinados sistemas de bases de datos en php.net.

9.1.3 Excepciones y opciones con PDO

PDO maneja los errores en forma de excepciones, por lo que la conexión siempre ha de ir encerrada en un bloque `try/catch`. Se puede (y se debe) especificar el modo de error estableciendo el atributo `error mode`:

```

1 | $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);
2 | $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
3 | $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

```

No importa el modo de error, si existe un fallo en la conexión siempre produce una excepción, por eso se conecta con `try/catch`.

- **PDO::ERRMODE_SILENT**

Es el modo de error por defecto. Si se deja así habrá que comprobar los errores de forma parecida a como se hace con `mysql`. Se tendrían que emplear `PDO::errorCode()` y `PDO::errorInfo()` o su versión en `PDOStatement` `PDOStatement::errorCode()` y `PDOStatement::errorInfo()`.

- **PDO::ERRMODE_WARNING**

Además de establecer el código de error, PDO emitirá un mensaje `E_WARNING`. Modo empleado para depurar o hacer pruebas para ver errores sin interrumpir el flujo de la aplicación.

- **PDO::ERRMODE_EXCEPTION**

Además de establecer el código de error, PDO lanzará una excepción `PDOException` y establecerá sus propiedades para luego poder reflejar el error y su información. Este modo se emplea en la mayoría de situaciones, ya que permite manejar los errores y a la vez esconder datos que podrían ayudar a alguien a atacar tu aplicación.

El modo de error se puede aplicar con el método `PDO::setAttribute` o mediante un array de opciones al instanciar PDO:

```

1 // Con un array de opciones
2 try {
3     $dsn = "mysql:host=localhost;dbname=$dbname";
4     $options = array( PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION );
5     $dbh = new PDO($dsn, $user, $password);
6 } catch (PDOException $e){
7     echo $e->getMessage();
8 }
9 // Con un el método PDO::setAttribute
10 try {
11     $dsn = "mysql:host=localhost;dbname=$dbname";
12     $dbh = new PDO($dsn, $user, $password);
13     $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
14 } catch (PDOException $e){
15     echo $e->getMessage();
16 }

```

Existen más opciones aparte de el modo de error ATTR_ERRMODE, algunas de ellas son:

- ATTR_CASE. Fuerza a los nombres de las columnas a mayúsculas o minúsculas (CASE_LOWER, CASE_UPPER).
- ATTR_TIMEOUT. Especifica el tiempo de espera en segundos.
- ATTR_STRINGIFY_FETCHES. Convierte los valores numéricos en cadenas.

9.1.4 Registrar datos con PDO

La clase PDOStatement es la que trata las sentencias SQL. Una instancia de PDOStatement se crea cuando se llama a PDO->prepare(), y con ese objeto creado se llama a métodos como bindParam() para pasar valores o execute() para ejecutar sentencias. PDO facilita el uso de sentencias preparadas en PHP, que mejoran el rendimiento y la seguridad de la aplicación. Cuando se obtienen, insertan o actualizan datos, el esquema es: PREPARE -> [BIND] -> EXECUTE. Se pueden indicar los parámetros en la sentencia con un interrogante ¿.º mediante un nombre específico. Utilizando interrogantes para los valores

```

1 // Prepare
2 $stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES
   (?, ?)");
3 // Bind
4 $nombre = "Peter";
5 $ciudad = "Madrid";
6 $stmt->bindParam(1, $nombre);
7 $stmt->bindParam(2, $ciudad);
8 // Execute
9 $stmt->execute();
10 // Bind
11 $nombre = "Martha";
12 $ciudad = "Cáceres";
13 $stmt->bindParam(1, $nombre);
14 $stmt->bindParam(2, $ciudad);
15 // Execute
16 $stmt->execute();

```

Utilizando variables para los valores

```
1 // Prepare
2 $stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES
   (:nombre,
3 // Bind
4 $nombre = "Charles";
5 $ciudad = "Valladolid";
6 $stmt->bindParam(':nombre', $nombre);
7 $stmt->bindParam(':ciudad', $ciudad);
8 // Execute
9 $stmt->execute();
10 // Bind
11 $nombre = "Anne";
12 $ciudad = "Lugo";
13 $stmt->bindParam(':nombre', $nombre);
14 $stmt->bindParam(':ciudad', $ciudad);
15 // Execute
16 $stmt->execute();
```

También existe un método lazy, que es pasando los valores mediante un array (siempre array, aunque sólo haya un valor) al método execute():

```
1 // Prepare:
2 $stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES
   (:nombre,
3 $nombre = "Luis";
4 $ciudad = "Barcelona";
5 // Bind y execute:
6 if($stmt->execute(array(':nombre'=>$nombre, ':ciudad'=>$ciudad))) {
7     echo "Se ha creado el nuevo registro!";
8 }
```

Es el método execute() el que realmente envía los datos a la base de datos.

Si no se llama a execute no se obtendrán los resultados sino un error.

Una característica importante cuando se utilizan variables para pasar los valores es que se pueden insertar objetos directamente en la base de datos, suponiendo que las propiedades coinciden con los nombres de las variables:

```
1 class Clientes {
2     public $nombre;
3     public $ciudad;
4     public function __construct($nombre, $ciudad){
5         $this->nombre = $nombre;
6         $this->ciudad = $ciudad;
7     }
8     // ....Código de la clase....
9 }
10 $cliente = new Clientes("Jennifer", "Málaga");
11 $stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES
   (:nombre,
12 if($stmt->execute((array) $cliente)){
13     echo "Se ha creado un nuevo registro!";
14 };
```

9.1.5 Diferencia entre bindParam() y bindValue()

Existen dos métodos para enlazar valores: bindParam() y bindValue(): Con bindParam() la variable es enlazada como una referencia y sólo será evaluada cuando se llame a execute():

```
1 // Prepare:
2 $stmt = $dbh->prepare("INSERT INTO Clientes (nombre) VALUES (:nombre)");
3 $nombre = "Morgan";
4 // Bind
5 $stmt->bindParam(':nombre', $nombre); // Se enlaza a la variable $nombre
6 // Si ahora cambiamos el valor de $nombre:
7 $nombre = "John";
8 $stmt->execute(); // Se insertará el cliente con el nombre John
```

Con bindValue() se enlaza el valor de la variable y permanece hasta execute():

```
1 // Prepare:
2 $stmt = $dbh->prepare("INSERT INTO Clientes (nombre) VALUES (:nombre)");
3 $nombre = "Morgan";
4 // Bind
5 $stmt->bindValue(':nombre', $nombre); // Se enlaza al valor Morgan
6 // Si ahora cambiamos el valor de $nombre:
7 $nombre = "John";
8 $stmt->execute(); // Se insertará el cliente con el nombre Morgan
```

En la práctica bindValue() se usa para insertar datos sólo una vez, y bindParam() cuando se tienen que pasar datos múltiples (desde un array por ejemplo).

Ambas funciones aceptan un tercer parámetro, que define el tipo de dato que se espera. Los data types más utilizados son: PDO::PARAM_BOOL (booleano), PDO::PARAM_NULL (null), PDO::PARAM_INT (integer) y PDO::PARAM_STR(string).

9.1.6 Consultar datos con PDO

La consulta de datos se realiza mediante PDOStatement::fetch, que obtiene la siguiente fila de un conjunto de resultados. Antes de llamar a fetch (o durante) hay que especificar como se quieren devolver los datos:

- PDO::FETCH_ASSOC: devuelve un array indexado cuyos keys son el nombre de las columnas.
- PDO::FETCH_NUM: devuelve un array indexado cuyos keys son números.
- PDO::FETCH_BOTH: valor por defecto. Devuelve un array indexado cuyos keys son tanto el nombre de las columnas como números.
- PDO::FETCH_BOUND: asigna los valores de las columnas a las variables establecidas con el método PDOStatement::bindColumn.
- PDO::FETCH_CLASS: asigna los valores de las columnas a propiedades de una clase. Crea las propiedades si éstas no existen.

- PDO::FETCH_INTO: actualiza una instancia existente de una clase.
- PDO::FETCH_OBJ: devuelve un objeto anónimo con nombres de propiedades que corresponden a las columnas.
- PDO::FETCH_LAZY: combina PDO::FETCH_BOTH y PDO::FETCH_OBJ, creando los nombres de las propiedades del objeto tal como se accedieron.

Los más utilizados son FETCH_ASSOC, FETCH_OBJ, FETCH_BOUND y FETCH_CLASS. Veamos un ejemplo de los dos primeros:

```

1 // FETCH_ASSOC
2 $stmt = $dbh->prepare("SELECT * FROM Clientes");
3 // Especificamos el fetch mode antes de llamar a fetch()
4 $stmt->setFetchMode(PDO::FETCH_ASSOC);
5 // Ejecutamos
6 $stmt->execute();
7 // Mostramos los resultados
8 while ($row = $stmt->fetch()){
9     echo "Nombre: {"$row["nombre"]} <br>";
10    echo "Ciudad: {"$row["ciudad"]} <br><br>";
11 }
12 // FETCH_OBJ
13 $stmt = $dbh->prepare("SELECT * FROM Clientes");
14 // Ejecutamos
15 $stmt->execute();
16 // Ahora vamos a indicar el fetch mode cuando llamamos a fetch:
17 while($row = $stmt->fetch(PDO::FETCH_OBJ)){
18     echo "Nombre: " . $row->nombre . "<br>";
19     echo "Ciudad: " . $row->ciudad . "<br>";
20 }

```

Con FETCH_BOUND debemos emplear el método bindColumn():

```

1 // Preparamos
2 $stmt = $dbh->prepare("SELECT nombre, ciudad FROM Clientes");
3 // Ejecutamos
4 $stmt->execute();
5 // bindColumn
6 $stmt->bindColumn(1, $nombre);
7 $stmt->bindColumn('ciudad', $ciudad);
8 while ($row = $stmt->fetch(PDO::FETCH_BOUND)) {
9     $cliente = $nombre . ": " . $ciudad;
10    echo $cliente . "<br>";
11 }

```

El estilo de devolver los datos FETCH_CLASS es algo más complejo: devuelve los datos directamente a una clase. Las propiedades del objeto se establecen ANTES de llamar al constructor. Si hay nombres de columnas que no tienen una propiedad creada para cada una, se crean como public. Si los datos necesitan una transformación antes de que salgan de la base de datos, se puede hacer automáticamente cada vez que se crea un objeto:

```

1 class Clientes {
2     public $nombre;
3     public $ciudad;

```

```

4 public $otros;
5 public function __construct($otros = ''){
6     $this->nombre = strtoupper($this->nombre);
7     $this->ciudad = mb_substr($this->ciudad, 0, 3);
8     $this->otros = $otros;
9 }
10 // ....Código de la clase....
11 }
12 $stmt = $dbh->prepare("SELECT * FROM Clientes");
13 $stmt->setFetchMode(PDO::FETCH_CLASS, 'Clientes');
14 $stmt->execute();
15 while ($objeto = $stmt->fetch()){
16     echo $objeto->nombre . " -> ";
17     echo $objeto->ciudad . "<br>";
18 }

```

Con lo anterior hemos podido modificar cómo queríamos mostrar nombre y ciudad de cada registro. A nombre lo hemos puesto en mayúsculas y de ciudad sólo hemos mostrado las tres primeras letras.

Si lo que quieres es llamar al constructor ANTES de que se asignen los datos, se hace lo siguiente:

```
$stmt->setFetchMode(PDO::FETCH_CLASS PDO::FETCH_PROPS_LATE, 'Clientes');
```

Si en el ejemplo anterior añadimos `PDO::FETCH_PROPS_LATE`, el nombre y la ciudad se mostrarán como aparecen en la base de datos.

También se pueden pasar argumentos al constructor cuando se quieren devolver datos en objetos con PDO: `$stmt->setFetchMode(PDO::FETCH_CLASS, 'Clientes', array('masdatos'));`

O incluso datos diferentes para cada objeto:

```

1 $i = 0;
2 while ($row = $stmt->fetch(PDO::FETCH_CLASS, 'Clientes', array($i))) {
3     // Código para hacer algo
4     $i++;
5 }

```

Finalmente, para la consulta de datos también se puede emplear directamente `PDOStatement::fetchAll()`, que devuelve un array con todas las filas devueltas por la base de datos con las que poder iterar. También acepta estilos de devolución:

```

1 // fetchAll() con PDO::FETCH_ASSOC
2 $stmt = $dbh->prepare("SELECT * FROM Clientes");
3 $stmt->execute();
4 $clientes = $stmt->fetchAll(PDO::FETCH_ASSOC);
5 foreach($clientes as $cliente){
6     echo $cliente['nombre'] . "<br>";
7 }
8 // fetchAll() con PDO::FETCH_OBJ
9 $stmt = $dbh->prepare("SELECT * FROM Clientes");
10 $stmt->execute();
11 $clientes = $stmt->fetchAll(PDO::FETCH_OBJ);
12 foreach ($clientes as $cliente){
13     echo $cliente->nombre . "<br>";
14 }

```


9.1.7 Diferencia entre query() y prepare()/execute()

En los ejemplos anteriores para las sentencias en PDO, no se ha introducido el método query(). Este método ejecuta la sentencia directamente y necesita que se escapen los datos adecuadamente para evitar ataques SQL Injection y otros problemas.

execute() ejecuta una sentencia preparada lo que permite enlazar parámetros y evitar tener que escapar los parámetros. execute() también tiene mejor rendimiento si se repite una sentencia múltiples veces, ya que se compila en el servidor de bases de datos sólo una vez.

Ya hemos visto como funcionan las sentencias preparadas con prepare() y execute(), vamos a ver un ejemplo con query():

```
1 $stmt = $dbh->query("SELECT * FROM Clientes");
2 $clientes = $stmt->fetchAll(PDO::FETCH_OBJ);
3 foreach ($clientes as $cliente){
4     echo $cliente->nombre . "<br>";
5 }
```

Se cambia prepare por query y se quita el execute.

9.1.8 Otras funciones de utilidad

Existen otras funciones en PDO que pueden ser de utilidad: PDO::exec(). Ejecuta una sentencia SQL y devuelve el número de filas afectadas. Devuelve el número de filas modificadas o borradas, no devuelve resultados de una secuencia SELECT:

```
1 // Si lo siguiente devuelve 1, es que se ha eliminado correctamente:
2 echo $dbh->exec("DELETE FROM Clientes WHERE nombre='Luis'");
3 // No devuelve el número de filas con SELECT, devuelve 0
4 echo $dbh->exec("SELECT * FROM Clientes");
```

PDO::lastInsertId(). Este método devuelve el id autoincrementado del último registro en esa conexión:

```
1 $stmt = $dbh->prepare("INSERT INTO Clientes (nombre) VALUES (:nombre)");
2 $nombre = "Angelina";
3 $stmt->bindValue(':nombre', $nombre);
4 $stmt->execute();
5 echo $dbh->lastInsertId();
```

PDOStatement::fetchColumn(). Devuelve una única columna de la siguiente fila de un conjunto de resultados. La columna se indica con un integer, empezando desde cero. Si no proporciona valor, obtiene la primera columna.

```
1 $stmt = $dbh->prepare("SELECT * FROM Clientes");
2 $stmt->execute();
3 while ($row = $stmt->fetchColumn(1)){
4     echo "Ciudad: $row <br>";
5 }
```

PDOStatement::rowCount(). Devuelve el número de filas afectadas por la última sentencia SQL:

```
1 $stmt = $dbh->prepare("SELECT * FROM Clientes");
2 $stmt->execute();
3 echo $stmt->rowCount();
```

9.1.9 Transacciones con PDO

Debido a que no todas las bases de datos soportan transacciones, PHP corre en el modo de auto-commit que ejecuta cada instrucción individual en forma implícita. Si se desea usar transacciones, y no se desea utilizar el modo de auto-commit, es necesario invocar el método `PDO::beginTransaction()` al inicio de la transacción. Si el manejador de la base de datos no permite el uso de transacciones se producirá una excepción (`PDOException`). Cuando se acabe de especificar la transacción se pueden utilizar los métodos `PDO::Commit` para aplicar dichas sentencias, o bien, `PDO::rollBack` para abortar dicha transacción.

```
1 <?php
2 try {
3     $dbh = new PDO('sqlite:test.db');
4     echo "Connected\n";
5 } catch (Exception $e) {
6     die("Unable to connect: " . $e->getMessage());
7 }
8
9 try {
10    $dbh->beginTransaction();
11    $dbh->exec("INSERT INTO countries (name, area, population, density)
12    values ('Belice ',22966,334000,14.54)");
13    $dbh->exec("INSERT INTO countries (name, area, population, density)
14    values ('Costa Rica ',51100,4726000,92.49)");
15    $dbh->exec("INSERT INTO countries (name, area, population, density)
16    values ('El Salvador ',21041,6108000,290.29)");
17    $dbh->exec("INSERT INTO countries (name, area, population, density)
18    values ('Guatemala ',108894,15284000,140.36)");
19    $dbh->exec("INSERT INTO countries (name, area, population, density)
20    values ('Honduras ',112492,8447000,75.09)");
21    $dbh->commit();
22 } catch (Exception $e) {
23    $dbh->rollBack();
24    echo "Failed: " . $e->getMessage();
25 }
26 ?>
```

Si una transacción no fue terminada con la instrucción `commit` y el programa finaliza, la base de datos abortará la transacción automáticamente. Cuando ejecutamos varias sentencias de vez, como `INSERT`, es preferible utilizar transacciones ya que agrupa todas las acciones y las revierte en caso de que haya algún error.

Una transacción en PDO comienza con el método `PDO::beginTransaction()`. Éste, desactiva cualquier otro `commit`, sentencia SQL o consultas que aún no son `committed` hasta que la transacción es `committed` con `PDO::commit()`. Cuando este método es llamado, todas las acciones pendientes se activan y la conexión a la base de datos vuelve de nuevo a su estado por defecto que es `auto-commit`. Con `PDO::rollback()` se revierten los cambios realizados durante la transacción.

```
1 try {
2     $dbh->beginTransaction();
3     $dbh->query("INSERT INTO Clientes (nombre, ciudad)
4     $dbh->query("INSERT INTO Clientes (nombre, ciudad)
5     $dbh->query("INSERT INTO Clientes (nombre, ciudad)
6     $dbh->query("INSERT INTO Clientes (nombre, ciudad)
```

```

7   $dbh->query("INSERT INTO Clientes (nombre, ciudad)
8   $dbh->commit();
9   echo "Se han introducido los nuevos clientes";
10  } catch (Exception $e){
11   echo "Ha habido algún error";
12   $dbh->rollback();
13  }

```

9.1.10 sentencias preparadas

Una instrucción preparada es un tipo de plantilla para SQL que puede ser personalizada utilizando parámetros. Existen dos beneficios de utilizar sentencias preparadas : la base de datos únicamente compilará una vez la instrucción lo cual ahorra mucho tiempo, y los parámetros no necesitan comillas ya que el manejador se encarga de agregarlas a la instrucción. El realizar el enlace (bind) de parámetros se puede realizar por mediante el nombre del parámetro o por posición (utilizando el símbolo ?).

```

1  <?php
2  try {
3      $dbh = new PDO('sqlite:test.db');
4      echo "Connected\n";
5  } catch (Exception $e) {
6      die("Unable to connect: " . $e->getMessage());
7  }
8
9  try {
10     $stmt = $dbh->prepare("INSERT INTO countries (name, area,
11                           population, density)
12                           VALUES (:name, :area, :population, :
13                                   density)");
14     $stmt->bindParam(':name', $name);
15     $stmt->bindParam(':area', $area);
16     $stmt->bindParam(':population', $population);
17     $stmt->bindParam(':density', $density);
18
19     $dbh->beginTransaction();
20     $name = 'Nicaragua'; $area = 129494; $population = 602800; $density
21         = 46.55;
22     $stmt->execute();
23     $name = 'Panama'; $area = 78200; $population = 3652000; $density =
24         46.70;
25     $stmt->execute();
26     $dbh->commit();
27 } catch (Exception $e) {
28     $dbh->rollBack();
29     echo "Failed: " . $e->getMessage();
30 }
31 ?>

```

Adicionalmente, es posible utilizar un arreglo para pasar los parámetros de la consulta. En este caso no es necesario incluir el enlace (bind) de parámetros. Es importante notar que el orden de los parámetros resulta vital aquí.

```

1 <?php
2 try {
3     $dbh = new PDO('sqlite:test.db');
4     echo "Connected\n";
5 } catch (Exception $e) {
6     die("Unable to connect: " . $e->getMessage());
7 }
8
9 try {
10     $stmt = $dbh->prepare("INSERT INTO countries (name, area,
11                             population, density)
12                             VALUES (?, ?, ?, ?)");
13     $dbh->beginTransaction();
14     $stmt->execute(array('Nicaragua', 129494, 602800, 46.55));
15     $stmt->execute(array('Panama', 78200, 3652000, 46.70));
16     $dbh->commit();
17 } catch (Exception $e) {
18     $dbh->rollBack();
19     echo "Failed: " . $e->getMessage();
20 }
21 }
22 ?>

```

9.1.11 Recuperación de datos

El método `PDOStatement::fetch` permite obtener la siguiente fila de un conjunto de resultados de una consulta. Esta instrucción tiene varios estilos de recuperación, entre ellos:

- 1 `PDO::FETCH_NUM`: Retorna la siguiente fila como un arreglo indexado por posición.
- 2 `PDO::FETCH_ASSOC`: Retorna la siguiente fila como un arreglo indexado por el nombre de la columna.
- 3 `PDO::FETCH_OBJ`: Retorna la siguiente fila como un objeto anónimo con los nombres de las columnas como propiedades.

Si se produce un error, la instrucción `fetch` retornará `FALSE`.

```

1 <html>
2 <?php
3 try {
4     $dbh = new PDO('sqlite:test.db');
5 } catch (Exception $e) {
6     die("Unable to connect: " . $e->getMessage());
7 }
8 try {
9     $sth = $dbh->prepare("SELECT * FROM countries");
10    $sth->execute();
11    echo "<table border=1>";
12    echo "<tr><th>Country</th><th>Area</th><th>People</th><th>Dens.</th></tr>";
13    while ($result = $sth->fetch(PDO::FETCH_ASSOC)) {
14        echo "<tr><td>". $result['name'] . "</td><td>". $result['area'] .

```

```

15         "</td><td>".$result['population']. "</td><td>".$result['
           density'].
16         "</td></tr>";
17     }
18     echo "</table>";
19 } catch (Exception $e) {
20     echo "Failed: " . $e->getMessage();
21 }
22 ?>
23 </html>

```

Por su parte la instrucción `PDOStatement::fetchAll` retornará un arreglo conteniendo todos las filas de un conjunto de resultados. El arreglo representa cada columna como un arreglo de valores por columnas o un objeto en donde las propiedades corresponden a los nombres de las columnas. Esta instrucción cuenta con varios modos al igual que la instrucción `fetch`, e inclusive se pueden especificar las columnas que se desean recuperar. Se retorna un arreglo vacío si no existen resultados, o `FALSE` si la consulta falla.

El siguiente ejemplo muestra el uso de la instrucción `fetchAll`, y al mismo tiempo se muestra una forma de recuperar los datos cuando no se conocen de antemano los nombres de las columnas ni la cantidad de ellas.

```

1 <html>
2 <?php
3     try {
4         $dbh = new PDO('sqlite:test.db');
5     } catch (Exception $e) {
6         die("Unable to connect: " . $e->getMessage());
7     }
8
9     try {
10         $sth = $dbh->prepare("SELECT * FROM countries");
11         $sth->execute();
12         echo "<table border=1><tr>";
13         $result = $sth->fetchAll(PDO::FETCH_ASSOC);
14         $keys = array_keys($result[0]);
15         foreach ($keys as $key)
16             echo "<th>".$key."</th>";
17         echo "</tr>";
18         foreach ($result as $row) {
19             echo "<tr>";
20             foreach ($keys as $key)
21                 echo "<td>".$row[$key]. "</td>";
22             echo "</tr>";
23         }
24         echo "</table>";
25     } catch (Exception $e) {
26         echo "Failed: " . $e->getMessage();
27     }
28     ?>
29 </html>

```

Capítulo 10

Un caso práctico

10.1 Caso de Estudio: Biblioteca Libraccio

La biblioteca Libraccio desea informatizar su operatoria básica en lo referente a: préstamos de ejemplares de libros a sus socios, las respectivas devoluciones de estos, y consultas acerca de la disponibilidad de los ejemplares. Los socios de la biblioteca pueden ser de 3 tipos: docente, lector y estudiante. Cada tipo de socio tiene diferentes condiciones de préstamo en cuanto a la duración y al número de ejemplares que puede retirar en préstamo. El número de días de suspensión, ante una devolución tardía de un ejemplar, también es diferente para cada tipo de socio. Cada libro tiene un isbn y un título, está escrito por uno o más autores, y es publicado por un editorial en una fecha de edición. Cada ejemplar de libro tiene un código único que lo identifica, y se conoce si está o no en mantenimiento por un eventual deterioro. Para la creación de este sistema de información se procede a:

1. realizar una visita de las instalaciones de la biblioteca para reunirnos con el personal encargado quienes proporcionarán la información necesaria en el proceso de prestamos y devoluciones de libros,
2. observar los procesos y cómo se almacena la información para la organización y control.

Teniendo en cuenta estos aspectos se identifica los requerimientos del sistema para las aplicaciones web y móvil.

- Reunión con el cliente para determinar requerimientos iniciales:
 - Documento de análisis.
 - Prototipo requisitos.
- Backlog:
 - Nuevos requerimientos del cliente.
 - Priorizados según sus necesidades.

Luego de obtener la información de las reuniones, se procede a su análisis para el diseño preliminar de la base de datos, que involucra la verificación con la biblioteca para su posterior implementación en un gestor de base de datos relacional.

Se diseña un mapa de navegación de la aplicación web y su construcción inicia con el desarrollo

de los módulos de registro, actualización y eliminación en la misma, para lo que se usará Symfony versión 3.2 que es un framework orientado PHP para crear sitios web y aplicaciones web, se utilizará FOSRestBundle para hacer APIs RESTful de forma más óptima para construir un controlador que funciona tanto para HTML, así como JSON / XML (Symfony, 2017).

Para el segundo entregable se crea el módulo de consultas a la información de la base de datos. Además, se crea la sección de aprobación o negación de las solicitudes de préstamo, y se podrá compartir en Facebook la información de cada animal de compañía.

10.1.1 Diseño de la interacción

Los documentos a entregar: diagrama de navegación y guión técnico.

10.1.1.1 Diagrama de flujo de la navegación

Es un documento que se utiliza para plasmar la estructura de la navegación del entorno. Nace, obviamente, del diagrama de contenidos, pero con la diferencia de que ahora se muestran en él no apartados, sino pantallas: si se prevé que un apartado del diagrama de contenidos requerirá más de una pantalla, ahora es el momento de plasmarlas.

La estructura ha de responder a las pantallas que se prevé diseñar. En este momento se hace un cálculo aproximado del número de pantallas que debe contener el producto, teniendo en cuenta las consideraciones hechas en el apartado de información en un producto digital.

Las ideas clave a tener en cuenta en la elaboración de un diagrama de flujo son:

Cada recuadro del diagrama de flujo debe corresponderse con una pantalla. Si prevemos que una sección será larga, debemos prever más de una pantalla.

Cada recuadro debe contener:

El título provisional de la sección a que responde.

Una referencia que retomamos luego en el guión técnico. La forma de referenciar es arbitraria y variada, pero es útil que la propia referencia indique el nivel jerárquico.

Es necesario considerar las relaciones entre las secciones. La unidireccionalidad o bidireccionalidad de una relación tiene implicaciones en la presentación porque requerirá un número determinado de iconos por página.

El diagrama de flujo representa la navegabilidad en nuestro producto y las relaciones entre las diferentes pantallas. Este documento se convertirá, pues, en el referente clave para los siguientes pasos y será el material previo a nuestro guión, como veremos en el apartado siguiente.

Aquí se presenta un diagrama de contenidos de un producto muy sencillo.

figura esquema.gif

10.1.1.2 Guión técnico

El guión técnico es un documento donde se diseñan cuidadosamente todas las pantallas del diagrama de flujo y donde se organizan todas las ideas y decisiones tomadas acerca del diseño.

En este documento se integran los contenidos del programa (texto, imágenes, animaciones, etc.) y se explicita el funcionamiento exacto de cada pantalla, es decir, la interactividad. Es un documento sumamente útil cuando el que diseña no es quien programa; en cualquier caso, es muy recomendable realizarlo siempre porque contribuye a la producción en:

- documentarla;
- planificar las posteriores fases.

La idea clave de un guión técnico es preparar la base para que el equipo de diseñadores y programadores puedan trabajar sobre un producto multimedia, y tiene una sola intención: integrar los contenidos que hemos preparado y el diseño de la interacción desarrollado en un solo esquema que muestre claramente cómo debe ser el material multimedia que estamos creando.

Ejemplo

Imaginemos que estamos trabajando en el apartado de recetas de nuestro ejemplo de cocina mediterránea:

Hemos coleccionado toda la información que necesitamos sobre recetas (actuales, clásicas, conocidas, poco conocidas), y las organizamos por temas siguiendo nuestro esquema de diseño.

Decidiremos qué apartados concretos tiene cada receta, porque en este caso el contenido se puede estructurar fácilmente en forma de fichas parecidas. Por ejemplo: ingredientes, utensilios, explicación paso a paso y vídeo del procedimiento.

Dibujaremos unos esbozos de cada pantalla en los que determinaremos los espacios que requerirá cada una de las pantallas de las recetas y los vínculos y relaciones con otras pantallas.

Partir de la creación de la estructura

En la fase anterior, el documento a entregar era un diagrama de flujo de los contenidos donde concretábamos la estructura de nuestro producto. Para empezar a crear nuestro guión necesitamos la estructura que hemos planteado.

figura 361_a.gif

Para realizar el guión técnico es aconsejable partir del diseño minucioso de algunas de las pantallas más importantes. Se trata de seleccionar las pantallas de forma que cumplan los siguientes requisitos:

Una pantalla de cada nivel.

Pantallas de las cuales dependan las demás.

Pantallas específicas, o bien por su contenido o bien por su interactividad (por ejemplo, la pantalla del chat).

Plantear la interacción entre estas pantallas clave nos ayudará a descubrir si la navegación presenta algún inconveniente.

Ejemplo

Por ejemplo, del diagrama de flujo del producto cocina mediterránea, hemos seleccionado las siguientes pantallas clave:

figura 361_b.gif

Qué debe contener un guión técnico

La apariencia de un guión técnico puede ser muy diversa según el tipo de programa que vayamos a desarrollar. En cualquier caso, un guión representa todas y cada una de nuestras pantallas y debe contener la información clave para cada una de ellas.

Cada una de las pantallas del diagrama de flujo se convertirá ahora en una ficha. De cada pantalla debemos exponer claramente:

Qué pantalla es: en el diagrama de contenidos utilizamos un título y una referencia que ahora debemos incluir en cada ficha.

Qué espacios tiene: esbozo de cada pantalla o una descripción lo más clara posible de lo que pretendemos que se muestre. Más que hacer un dibujo detallado, es imprescindible definir los espacios que ocupará cada uno de los elementos de la pantalla.

Qué se lee: el texto que aparece en pantalla (si lo hay). El guión técnico debe contener el texto definitivo escrito de forma adecuada y revisado y corregido tal y como se programará en la pantalla.

Qué se ve: las imágenes, animaciones o vídeos que tiene esta pantalla.

Qué se oye: sonido, voz en off, música, efectos especiales, etc.

Qué interactividad tiene: qué elementos serán activos y cómo reaccionarán a la acción del usuario. Debe quedar claro cómo se mueve el usuario de ésta a otras pantallas, de qué enlaces dispone y las relaciones con las demás pantallas, cuando las haya.

Para facilitar la lectura del guión técnico, estos elementos se agrupan de forma que se facilite la ubicación de todos ellos en una única página o, a lo sumo, en dos páginas (una para el esbozo y otra para los elementos y la interacción). Una distribución posible es la siguiente:

figura plantilla.gif

En este planteamiento, cumplimentar cada ficha requiere referenciar cada elemento:

El esbozo de la pantalla contiene recuadros que reservarán espacios a los diferentes elementos que compondrán la pantalla. Cada recuadro debe disponer también de una referencia.

Comentario del autor

Es útil que estas referencias denoten el tipo de elemento al que refieren: así, por ejemplo, los recuadros que contienen imágenes podrían referenciarse como I1, I2, I3..., los de texto T1, T2..., y los de animaciones como A1, A2...

También podría buscarse una nomenclatura más compleja que relacionara cada elemento con la pantalla en la que se inserta. Así, si la referencia de la pantalla fuera P3.3.4, las imágenes podrían ser I334a, I334b...

Los elementos (textos, imágenes, animaciones, sonidos, etc.) que se han referenciado en el esbozo deben aparecer en su recuadro respectivo. En función de la extensión requerida, se puede trabajar en el propio recuadro o en un documento aparte que se presentará anexado al guión técnico.

En el recuadro del propio guión En un documento aparte anexado Elementos textuales Insertar el texto completo a continuación de la referencia de forma que el recuadro de texto contenga todo el contenido completo.

T77: La sopa de ajo es un plato que se sirve tibio y se acompaña de... Anexar en un documento aparte los textos y, en el guión técnico, especificar en qué apartado de qué archivo se encuentra el texto de esta pantalla.

T54: ver Anexo.doc apartado T54. Imágenes Sonidos Describir qué imágenes se están buscando. Se está pidiendo a los programadores que las busquen.

I342: Perspectiva global de una mesa vestida informalmente en la que se ha servido una sopa de ajo acompañada de un vino tinto. Anexar un soporte digital que contenga las imágenes organizadas. En el guión se debe indicar el nombre del fichero y en qué carpeta se halla.

I345: ver CD Imagenes/cap3/i345.jpg Animaciones Describir la acción de la animación (es recomendable si es una acción corta).

A9: Una olla contiene poco caldo y algunas verduras. Unas manos inclinan la olla y una batidora bate el caldo y las verduras a potencia 4. Anexar un documento que contenga el guión completo de cada una de las animaciones.

A8: ver Animaciones.doc

La interactividad entre pantallas y entre los elementos de la propia pantalla. La interactividad se puede expresar de muchas formas:

I2, al hacer clic, conduce a P3.3

I2 (clic) ->P3.3

I2 (onmouseover) ->I3 (onmouseout) ->I2

T4 (clic) ->sonido_campana.mp3

Ejemplo

En el ejemplo siguiente, perteneciente al producto de cocina mediterránea, se optó por añadir una descripción de la pantalla y por diferenciar la navegación entre pantallas de la interacción de los elementos de la propia pantalla.

figura 363_a.gif

Crear un guión técnico a medida

En los ejemplos siguientes se muestran guiones de materiales orientados a la información que mantienen esquemas similares, pero cada uno de ellos ha sido construido según las características propias de sus pantallas.

figura 364_a.gif

Ejemplo de pantalla de guión del programa «La Torre más alta del ciberespacio» del producto Educalia. <http://www.educalia.com>

figura 364_c.gif

Pantalla del guión del proyecto FUGA (Formación Útil para Gente Autoinsuficiente), creado por los alumnos de Comunicación Audiovisual de la Universidad de Barcelona.

figura 364_f.gif

Ejemplo de una pantalla de guión del producto «Activa Multimedia» de Plaza&Janés. Una aventura gráfica con un nivel de programación complejo. El guión incluye continuamente referencias gráficas para cada pantalla y descripciones concretas de todas las acciones a programar en cada una de ellas.

figura 364_g.gif

Ejemplo del guión del producto «Formación a Distancia do Profesorado» de la Xunta de Galicia, para la formación a distancia de maestros de las escuelas rurales de Galicia. Todo el programa se basa en una colección de actividades lúdicas sobre los aspectos de la ley de enseñanza de la LOGSE. Todas las actividades se presentan para la revisión y el refuerzo de los contenidos aprendidos con los materiales.

Cómo evitar pantallas repetitivas en el guión

En productos muy grandes, se acostumbran a encontrar pantallas que se asemejan mucho unas a otras. Evitad realizar tareas extremadamente rutinarias o repetitivas.

Si debéis crear cincuenta pantallas similares, no es necesario elaborar cincuenta fichas iguales. Buscad la forma de simplificar el guión: quizá la solución sea tener una ficha por cada pantalla tipo.

Si hay una parte común a una larga serie de pantallas (como por ejemplo un menú de navegación) es recomendable desmembrar el menú y tratarlo diferenciadamente en una ficha aparte del guión; de esta manera, simplificaremos las referencias necesarias en el recuadro de interactividad del resto de pantallas.

Ejemplo

Todas las pantallas secundarias del producto de cocina mediterránea se basan en un diseño que contiene unos menús de navegación en la parte lateral izquierda y en la parte superior.

figura 362_a.gif

Para evitar tener que indicar la navegación e interacción de estos menús en cada una de las fichas del guión, escogemos dedicar una ficha a estos menús y en las restantes sólo nos fijaremos en los contenidos centrales.

figura 363_b.gif

Revisión y evaluación del guión

Una vez realizado el guión conviene detenerse para revisar el diseño antes de proceder a la producción. Debemos plantearnos si:

- la interfaz diseñada es la correcta;

- los contenidos son suficientes y no demasiado extensos;

- los contenidos están bien escritos, son comprensibles para los usuarios, son claros y concisos;

- los objetivos planteados inicialmente se consiguen con nuestro diseño;

- la estructura diseñada permite moverse con facilidad y lógica por el entorno.

Se trata de mirar atrás y asegurarnos que no olvidamos nada y que nuestro diseño es realmente válido. Realizar ahora la revisión y evaluar el diseño nos evita tener que cambiar el producto cuando esté programado.

Inicio

Bibliografía

- [1] GILLILLAN IAN. (2003), *La Biblia de MySQL. Anaya Multimedia.*
- [2] MORA.S. (S.F.), *Desarrollo de aplicaciones web.*
- [3] POTENCIER, Z. (2012), *Symfony la guía definitiva.*
- [4] PRESSMAN R. S. (2010), *Ingeniería de software. Un enfoque práctico. Mexico: McGraw-Hill.*
- [5] HANNA PHIL. (S.F.), *JSP Manual de Referencia, McGraw-Hill*
- [6] DARNEL RICK. (S.F.), *Java Script Quick Reference*
- [7] KITCHENHAM B. (S.F.), *Procedures for performing systematic reviews, Keele, UK, Keele Univ., vol. 33, no. TR/SE-0401, p. 28, 2004.*
- [8] BIOLCHINI J., P. G. MIAN, A. CANDIDA, AND C. NATALI, (2005), *Systematic Review in Software Engineering,” Engineering, vol. 679, no. May, pp. 1–31*
- [9] CALIDAD D. (2005), *Los Web Services y las características de calidad.*
- [10] IBM - ARQUITECTURA ORIENTADA A SERVICIOS (SOA) DE IBM - ESPAÑA.[ONLINE], <https://www-01.ibm.com/software/es/solutions/soa/>
- [11] M. N. MENDES E, (2006), *Web Engineering, Heidelberg: Springer Verlag*
- [12] MURUGESAN S. D. Y., (2001), *Web Engineering, Managing Diversity and Complexity of Web Application Development, Lecture Notes in Computer Science*
- [13] PRESSMAN R. Y L. DAVID L. (2010), *Web engineering : a practitioner’s approach, New York: McGraw-Hil*
- [14] RODRÍGUEZ A. N. (2009), *Metodologías del diseño usadas en Ingeniería Web, su vinculación con las NTICS, La Plata, Universidad Nacional de La Plata, Facultad de Informática*
- [15] GÓMEZ, J. M. (2012), *Aplicación de la Ingeniería Web a sitios Web, Universidad Autónoma de Baja California Sur*
- [16] DE MUYNCK, W. (2000), *Bridging the Gap between XML and Hypermedia: a Layered Transformational Approach, Tesis. Approach, Vrije Universiteit Brussel, Belgium.*
- [17] SCHWAVE, D. AND G. ROSSI, G. (1998), *An Object Oriented Approach to Web-Based Application Desing. En: Theory and Practice of Object Systems (TAPOS)*

- [18] SCHWAVE, D (2007) ET AL. (S.F.), *Engineering Web Applications for Reuse. IEEE Multi-media, Vol 8 Nro 1, pp 20-31.*
- [19] G. ROSSI; D. SCHWAVE AND FERNANDO LYARDET (S.F.), *Web application models are more than conceptual models. En: Proceedings of the First International Workshop on Conceptual Modeling and the WWW, Paris, France, November 1999.*
- [20] D. COWAN AND C. LUCENA. (1995), *Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse. IEEE Transactions on Software Engineering. Vol. 21, No. 3.*
- [21] N. KOCH, N. (2000), *Comparing Development Methods for Web Applications. Ludwig-Maximilians-University Munich, Institute of Computer Science Oettingenstr. 67, 80538 München, Germany.*
- [22] F. GARZOTTO; L. MAINETTI AND P. PAOLINI (1995), *Hypermedia design analysis. Communications of the ACM, 8(38), 74-86. 1995.*
- [23] ISAKOWITZ, T., STOHR, E. AND BALASUBRAMANIAN, P. (1995), *A methodology for the design of structured hypermedia applications. Communications of the ACM, 8(38), 34-44.*
- [24] D. LANGE (1996), *An object-oriented design approach for developing hypermedia information systems. Journal of Organizational Computing and Electronic Commerce, 6(3), 269-293. 1996.*
- [25] H. LEE; C. LEE AND C. YOO (1998), *A scenario-based object-oriented methodology for developing hypermedia information systems. En: Proceedings of 31st Annual Conference on Systems Science, Eds. Sprague R. 1998.*
- [26] DE TROYER, O. AND LEUNE, C. (1997), *WSDM: A user-centered design method for Web sites. En: Proceedings of the 7th International World Wide Web Conference*
- [27] CONALLEN. J. (1999), *Building Web application with UML. Addison Wesley.*
- [28] RUMBAUGH J., JACOBSON I. AND BOOCH G. (1999), *The Unified Modeling Language – Reference Manual. Addison Wesley Longman, Inc.*
- [29] RUMBAUGH J., JACOBSON I. AND BOOCH G. (1999), *The Unified Modeling Language – User Guide. Addison Wesley Longman, Inc.*
- [30] SCHNEIDER, G. AND J. WINTERS, J. (2001), *Applying Use Cases: A Practical Guide. Second Edition. Addison Wesley*
- [31] DOMÍNGUEZ CH. JORGE,(1996), *Sistemas de Información Orientado a Objetos, IEASS, Editores, Venezuela.*
- [32] DYBA,TORRE; DINGSOYR TORGEIR . (2009), *What do we know about agile software development? - IEEE Software. <http://dx.doi.org/10.1109/MS.2009.145>*
- [33] MARTIN FOWLER, (2005), *The new methodology, <http://martinfowler.com/articles/newMethodology.htm>*
- [34] ALFREDO WEITZENFELD (2005), *Ingeniería de Software orientada a objetos con UML Java e Internet, Thomson editores, ISBN 970-686-190-4*

- [35] WINSTON W ROYCE, (1970), *Managing the development of large software systems*, *Proceedings, IEEE Wescon*, www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf
- [36] BARRY W BOEHM, (1988), *A Spiral Model of Software Development and Enhancement Computer, (IEEE) Volume 21, Issue 5, 1988.*
- [37] KENT BECK, DAVE CLEAL, (1999), *Optional Scope Contracts*
- [38] BILL VENNERS (2003), *Collective Ownership of Code and Text* , *A Conversation with Ward Cunningham, Artima Developer* , December 1, 2003, <http://www.artima.com/intv/ownership.html>
- [39] ROBERT L. NORD, JAMES E. TOMAYKO, ROB WOJCIK (2004), *Integrating Software-Architecture-Centric Methods into Extreme Programming*, *CMU/SEI-2004-TN-036, SEI*

Acerca del autor

Graduado en Física, Facultad de Ciencias, Universidad Nacional Autónoma de México (UNAM), Doctor en Ciencias de la Computación, Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas (IIMAS), UNAM.

Especialista en Economía Matemática, Centro de Investigación y Docencia Económica, CIDE, México. Cursante del Doctorado en Minería de Datos, Modelos y Sistemas Expertos por la Universidad de Illinois en Urbana-Champaigns (USA).

Ha sido profesor en la Facultad de Química y en la Facultad de Ingeniería, UNAM. Fue profesor en el Departamento de Ciencias Básicas, Universidad de Las Américas en Cholula, (UDLAP), Puebla, México y, durante seis años, profesor visitante en la Universidade Federal de Rio Grande do Sul (Brasil), profesor y jefe de sistemas postgrados de agronomía y veterinaria, Universidad Central de Venezuela (UCV), actualmente es profesor del Departamento de informática y del Departamento de Postgrado, Universidad Politécnica Territorial de Aragua (UPT Aragua), Venezuela.

Expositor y conferencista a nivel nacional e internacional.

Es asesor en mejora de procesos, gestión de proyectos, desarrollo de software corporativo en los sectores de servicios, banca, industria y gobierno.

El Dr. Domínguez es un especialista reconocido en base de datos, desarrollo de software y servidores en el área del software libre, así como un experto en LINUX DEBIAN.

En la actualidad orienta su trabajo a la creación y desarrollo de equipos de software de alto desempeño.

Autor de múltiples artículos y libros sobre la materia.

