|  |  |
| --- | --- |
|  | **2012** |
|  | **Benemérita Universidad Autónoma de Puebla****Facultad de Ciencias de la Computación** |

|  |
| --- |
| **[Aplicaciones web]** |
| M.C Pedro Bello López |

[Introducción 3](#_Toc313552904)

[1. Arquitectura y Servicio Internet 4](#_Toc313552905)

[1.1 Evolución de la arquitectura cliente servidor 5](#_Toc313552906)

[1.2 El Cliente y el Servidor 6](#_Toc313552907)

[1.3 Distribución entre el Cliente y el Servidor 8](#_Toc313552908)

[1.4 Características del modelo cliente Servidor 10](#_Toc313552909)

[1.5 Ventajas del Modelo Cliente Servidor 10](#_Toc313552910)

[1.6 Inconvenientes del Modelo Cliente Servidor 11](#_Toc313552911)

[2. Sentencias básicas en PHP 13](#_Toc313552912)

[2.1 Entorno de desarrollo de PHP 14](#_Toc313552913)

[2.2 Sintaxis Básica 14](#_Toc313552914)

[2.3 Variables y Operadores 16](#_Toc313552915)

[2.4 Sentencias de Control 18](#_Toc313552916)

[2.5 Tablas 20](#_Toc313552917)

[3. Funciones 23](#_Toc313552918)

[3.1 Llamado de scripts 23](#_Toc313552919)

[3.3 Clases en PHP 26](#_Toc313552920)

[3.4 Envío de Mensajes 29](#_Toc313552921)

[4. Formularios HTML 31](#_Toc313552922)

[4.1 Creando Formularios 31](#_Toc313552923)

[4.2 Entrada básica de datos 32](#_Toc313552924)

[5. Base de Datos con PHP 38](#_Toc313552925)

[5.1 SQL 38](#_Toc313552926)

[5.2. Comandos Básicos de SQL 39](#_Toc313552927)

[5.3 MySQL 44](#_Toc313552928)

[5.3.1 Comandos DDL 45](#_Toc313552929)

[5.4 Comandos DML 48](#_Toc313552930)

[5.5 Conexión de MySQL con PHP 50](#_Toc313552931)

[Bibliografía 52](#_Toc313552932)

# Introducción

La tecnología Cliente/Servidor, es un modelo que implica productos y servicios enmarcados en el uso de la tecnología de punta, y que permite la distribución de la información en forma ágil y eficaz a las diversas áreas de una organización (empresa o institución pública o privada), así como también fuera de ella. Con el crecimiento de las redes de computadoras esta propuesta Cliente Servidor se ha desarrollado enormemente.

Existen diversos puntos de vista sobre la manera en que debería efectuarse el procesamiento de datos, aunque la mayoría que opina, coincide en que nos encontramos en medio de un proceso de evolución que se prolongará todavía por algunos años y que cambiará la forma en que obtenemos y utilizamos la información almacenada electrónicamente.

El principal motivo detrás de esta evolución es la necesidad que tienen las organizaciones, de realizar sus operaciones más eficientemente, debido a la creciente presión competitiva a la que están sometidas, a la necesidad de que se reduzcan los costos y gastos de operación, al mismo tiempo que se generan productos y servicios más rápidamente y con mejor calidad.

En este contexto, es necesario establecer una infraestructura de procesamiento de información, que cuente con los elementos requeridos para proveer información adecuada, exacta y oportuna en la toma de decisiones y para proporcionar un mejor servicio a los clientes.

El modelo Cliente/Servidor reúne las características necesarias para proveer esta infraestructura, independientemente del tamaño y complejidad de las operaciones de las organizaciones públicas o privadas y, consecuentemente desempeña un papel importante en este proceso de evolución.

Un elemento básico e imprescindible en un modelo Cliente Servidor, es el uso de las bases de datos, por lo que es necesario el conocer una arquitectura que en este momento es una de las más importantes y utilizadas en el ámbito de enviar y recibir información, también es una herramienta potente para guardar los datos en una base de datos como servidor.

# 1. Arquitectura y Servicio Internet

El modelo Cliente Servidor, es un modelo para construir sistemas de información, que se sustenta en la idea de repartir el tratamiento de la información y los datos por todo el sistema informático, permitiendo mejorar el rendimiento del sistema global de información

Segun IBM: “Es la tecnología que proporciona al usuario final el acceso transparente a las aplicaciones, datos, servicios de cómputo o cualquier otro recurso del grupo de trabajo y/o, a través de la organización, en múltiples plataformas. El modelo soporta un medio ambiente distribuido en el cual los requerimientos de serviciohechos por estaciones de trabajo inteligentes o "clientes'', resultan en un trabajo realizado por otros computadores llamados servidores"

Desde el punto de vista de la arquitectura:

La arquitectura Cliente/Servidor agrupa conjuntos de elementos que efectúan procesos distribuidos y computo cooperativo.



Figura. 1.1 Esquema general de un servicio en el modelo Cliente Servidor

**Beneficios de la arquitectura Cliente/Servidor:**

1. Mejor aprovechamiento de la potencia de cómputo (Reparte el trabajo).
2. Reduce el tráfico en la Red. (Viajan requerimientos).
3. Opera bajo sistemas abiertos.
4. Permite el uso de interfaces gráficas variadas y versátiles.



Figura 1.2 Solicitud de una petición en la arquitectura Cliente/Servidor

## ****1.1 Evolución de la arquitectura cliente servidor****

****La era de la computadora central****

“Desde sus inicios el modelo de administración de datos a través de computadoras se basaba en el uso de terminales remotas, que se conectaban de manera directa a una computadora central”. Dicha computadora central se encargaba de prestar servicios caracterizados por que cada servicio se prestaba solo a un grupo exclusivo de usuarios.

**La era de las computadoras dedicadas**

Esta es la era en la que cada servicio empleaba su propia computadora que permitía que los usuarios de ese servicio se conectaran directamente. Esto es consecuencia de la aparición de computadoras pequeñas, de fácil uso, más baratas y más poderosas de las convencionales.

**La era de la conexión libre**

Hace mas de 10 años que la computadoras escritorio aparecieron de manera masiva. Esto permitió que parte apreciable de la carga de trabajo de cómputo tanto en el ámbito de cálculo como en el ámbito de la presentación se lleven a cabo desde el escritorio del usuario.

En muchos de los casos el usuario obtiene la información que necesita de alguna computadora de servicio. Estas computadoras de escritorio se conectan a las computadoras de servicio empleando software que permite la emulación de algún tipo de terminal. En otros de los casos se les transfiere la información haciendo uso de recursos magnéticos o por trascripción.

****La era del cómputo a través de redes****

Esta es la era que esta basada en el concepto de redes de computadoras, en la que la información reside en una o varias computadoras, los usuarios de esta información hacen uso de computadoras para laborar y todas ellas se encuentran conectadas entre si. Esto brinda la posibilidad de que todos los usuarios puedan acceder a la información de todas las computadoras y a la vez que los diversos sistemas intercambien información.

### ****La era de la arquitectura cliente servidor****

“En esta arquitectura la computadora de cada uno de los usuarios, llamada cliente, produce una demanda de información a cualquiera de las computadoras que proporcionan información, conocidas como servidores” estos últimos responden a la demanda del cliente que la produjo.

Los clientes y los servidores pueden estar conectados a una red local o una red amplia, como la que se puede implementar en una empresa o a una red mundial como lo es la Internet.

Bajo este modelo cada usuario tiene la libertad de obtener la información que requiera en un momento dado proveniente de una o varias fuentes locales o distantes y de procesarla como según le convenga. Los distintos servidores también pueden intercambiar información dentro de esta arquitectura.

## 1.2 El Cliente y el Servidor

La base fundamental de este modelo radica en el cliente (usuario) y el proveedor de servicios (Servidor).



Figura 1.3. Esquema el modelo Cliente Servidor

**El Cliente**: Es el que inicia unrequerimiento de servicio. El requerimientoinicial puede convertirse en múltiples requerimientos de trabajo a través de redes LAN o WAN. La ubicación de los datos o de las aplicaciones estotalmente transparente para el cliente.

**Características del cliente:**

* El Cliente oculta al Servidor y la Red.
* Detecta e intercepta peticiones de otras aplicaciones y puede redireccionarlas.
* Dedicado a la cesión del usuario ( Inicia...Termina ).
* El método más común por el que se solicitan los servicios es a través de RPC (Remote Procedure Calls).

**Funciones Comunes del Cliente:**

* Mantener y procesar todo el dialogo con el usuario.
* Manejo de pantallas.
* Menús e interpretación de comandos.
* Entrada de datos y validación.
* Procesamiento de ayudas.
* Recuperación de errores.

**El Servidor:** Es cualquier recurso de cómputo dedicado a responder alos requerimientos del cliente. Los servidores pueden estar conectadosa los clientes a través de redes LANs o WANs, para proveer de múltiples servicios a los clientes y ciudadanos tales como impresión, acceso a bases de datos, fax, procesamiento de imágenes, etc.

**Tipos Comunes de Servidores:**

* Servidor de Archivos (FTP, Novell).
* Servidor de Bases de Datos (SQL, CBASE, ORACLE, INFORMIX).
* Servidor de Comunicaciones
* Servidor de transacciones
* Servidor de Groupware
* Servidores de objetos
* Servidor Web
* Servidor de Impresión.
* Servidor de Terminal.
* Servidor de Aplicaciones (Windows NT, Novell).

**Funciones Comunes del Servidor:**

* Acceso, almacenamiento y organización de datos.
* Actualización de datos almacenados.
* Administración de recursos compartidos.
* Ejecución de toda la lógica para procesar una transacción.
* Procesamiento común de elementos del servidor (Datos, capacidad de CPU, almacenamiento en disco, capacidad de impresión, manejo de memoria y comunicación).

## 1.3 Distribución entre el Cliente y el Servidor

Dentro de los procesos que son manejados en una arquitectura Cliente/Servidor, se tiene 3 elementos básicos que deben de distribuirse entre el cliente y el servidor, estos elementos son:

* El manejo de Datos.
* La aplicación.
* La presentación.



Figura 1.4. Elementos básicos entre el Cliente y el Servidor

En la arquitectura Cliente/Servidor podemos distinguir los elementos que caracterizan dicha arquitectura, es decir:

* Puestos de Trabajo (Estaciones de trabajo o Clientes)
* Comunicaciones
* Servidores



Figura 1.5 Componentes de la arquitectura Cliente Servidor

**El Puesto de Trabajo o Cliente**: Una Estación de trabajo o microcomputador (PC: Computador Personal) conectado a una red, que le permite acceder y gestionar una serie de recursos» el cual se perfila como un puesto de trabajo universal. Nos referimos a un microcomputador conectado al sistema de información y en el que se realiza una parte mayoritaria de los procesos

**Los Servidores o Back-end:** Una máquina que suministra una serie de servicios como Bases de Datos, Archivos, Comunicaciones, etc).

Los Servidores, según la especialización y los requerimientos de los servicios que debe suministrar pueden ser:

* Mainframes
* Miniordenadores
* Especializados (Dispositivos de Red, Imagen, etc.)

Una característica a considerar es que los diferentes servicios, según el caso, pueden ser suministrados por un único Servidor o por varios Servidores especializados.

**Las Comunicaciones en sus dos vertientes**:

* Infraestructura de redes

Componentes Hardware y Software que garantizan la conexión física y la transferencia de datos entre los distintos equipos de la red.

* Infraestructura de comunicaciones

Componentes Hardware y Software que permiten la comunicación y su gestión, entre los clientes y los servidores.

La arquitectura Cliente/Servidor es el resultado de la integración de dos culturas. Por un lado, la del Mainframe que aporta capacidad de almacenamiento, integridad y acceso a la información y, por el otro, la del computador que aporta facilidad de uso (cultura de PC), bajo costo, presentación atractiva (aspecto lúdico) y una amplia oferta en productos y aplicaciones.

## 1.4 Características del modelo cliente Servidor

1. El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.
2. Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma.
3. Un servidor da servicio a múltiples clientes en forma concurrente.
4. Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los Clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.

## 1.5 Ventajas del Modelo Cliente Servidor

El esquema Cliente/Servidor posee las siguientes ventajas:

1. Uno de los aspectos que más ha promovido el uso de sistemas Cliente/Servidor, es la existencia de plataformas de hardware cada vez más baratas. Esto porque se pueden utilizar componentes, tanto de hardware como de software, de varios fabricantes, lo cual contribuye considerablemente a la reducción de costos y favorece la flexibilidad en la implantación y actualización de soluciones.
2. El esquema Cliente/Servidor facilita la integración entre sistemas diferentes y comparte información permitiendo, por ejemplo que las máquinas ya existentes puedan ser utilizadas pero utilizando interfaces mas amigables al usuario.
3. Con el uso de interfaces gráficas para el usuario, el esquema Cliente/Servidor presenta la ventaja, con respecto a uno centralizado, de que no es siempre necesario transmitir información gráfica por la red pues esta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red.
4. En el esquema Cliente/Servidor es que es más rápido el mantenimiento y el desarrollo de aplicaciones, pues se pueden emplear las herramientas existentes (por ejemplo los servidores de SQL o las herramientas de más bajo nivel como los sockets o el RPC ).
5. La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.
6. El esquema Cliente/Servidor contribuye además, a proporcionar, a los diferentes departamentos de una organización, soluciones locales, pero permitiendo la integración de la información relevante a nivel global.

## 1.6 Inconvenientes del Modelo Cliente Servidor

El esquema Cliente/Servidor tiene algunos inconvenientes que se mencionan a continuación:

1. Se cuenta con muy escasas herramientas para la administración y ajuste del desempeño de los sistemas.
2. En el desarrollo de aplicaciones Cliente/Servidor se deben tener en cuenta diferentes aspectos, como Redes, Lenguajes de Programación para Web, Multimedia, Bases de Datos, Sistemas operativos, Seguridad en Red, etc.
3. Es importante que los clientes y los servidores utilicen el mismo mecanismo (por ejemplo sockets o RPC), lo cual implica que se deben tener mecanismos generales que existan en diferentes plataformas.
4. Además, hay que tener estrategias pare el manejo de errores y para mantener la consistencia de los datos. La seguridad de un esquema Cliente/Servidor es otra preocupación importante. Por ejemplo, se deben hacer verificaciones en el cliente y en el servidor. También se puede recurrir a otras técnicas como el encriptamiento.
5. El desempeño es otro de los aspectos que se deben tener en cuenta en el esquema Cliente/Servidor. Problemas de este estilo pueden presentarse por congestión en la red, dificultad de tráfico de datos, etc.
6. Un aspecto directamente relacionado con lo anterior es el de cómo distribuir los datos en la red. En el caso de una organización, por ejemplo, éste puede ser hecho por departamentos, geográficamente, o de otras maneras. Hay que tener en cuenta que en algunos casos, por razones de confiabilidad o eficiencia, se pueden tener datos replicados, y que puede haber actualizaciones simultáneas.

En conclusión el modelo Cliente Servidor puede incluir múltiples plataformas, bases de datos, redes y sistemas operativos. Estos pueden ser de distintos proveedores, en arquitecturas propietarias y no propietarias y funcionando todos al mismo tiempo. Por lo tanto, su implantación involucra diferentes tipos de estándares: APPC, TCP/IP, OSI, NFS, DRDA corriendo sobre DOS, OS/2, Windows o PC UNIX, en Token­Ring, Ethernet, FDDI o medio coaxial, sólo por mencionar algunas de las posibilidades.

# 2. Introducción a MySQL

Una base de datos es una colección de datos clasificados y estructurados que son guardados en uno o varios ficheros pero referenciados como si de un único fichero se tratara. Para crear y manipular bases de datos relacionales, existen en el mercado varios sistemas administradores de bases de datos, por ejemplo: Access, SQL Server, Oracle y DB2. Otros sistemas administradores de bases de datos de interés y de libre distribución son MySQL y PostgreSQL. En este sentido MySQL es un sistema administrador de bases de datos que cubre con sobra las necesidades de un buen DBMS.

Los datos de una base de datos relacional se almacenan en tablas lógicamente relacionadas entre sí utilizando campos clave comunes. A su vez, cada tabla dispone los datos en *filas y columnas*. Por ejemplo, piense en el caso de una agenda de teléfonos. Los datos relativos a una agenda (nombre, dirección, teléfono, etc) son columnas que agrupamos en una fila. El conjunto de todas las filas de todos los teléfonos forman una tabla de la base de datos.

|  |  |  |
| --- | --- | --- |
| Nombre | Dirección | Teléfono |
| Aguado Rodríguez Jesús | Las Rablas 3, Barcelona | 932345678 |
| Cuesta Suñer, Ana María | Mayor 22, Madrid | 918765432 |
| … | … | … |

Como se puede observar, una tabla es una colección de datos presentada en forma de una matriz bidimensional, donde las filas reciben también el nombre de *tuplas* o *registros* y las columnas *campos*.

Los usuarios de un sistema administrador de bases de datos puede realizar sobre una determinada base operaciones como insertar, recuperar, modificar y eliminar datos, así como añadir nuevas tablas o eliminarlas. Estas operaciones se expresan generalmente en un lenguaje denominado SQL.

## 2.1 SQL

SQL es un lenguaje estándar para interactuar con bases de datos relacionales y es soportado prácticamente por todos los sistemas administradores de bases de datos actuales. En él, las unidades básicas son *tablas, columnas* y *filas*. La *tabla* proporciona una forma simple de relacionar los datos que componen la misma, una *columna* representa un dato presenta en la tabla, mientras que la *fila* representa un registro o entrada de la tabla.

## 2.2. Comandos Básicos de SQL

A continuación se presentan las operaciones más comunes para acceso a bases de datos incluyendo operaciones tanto de definición (CREATE) como de manipulación de datos (INSERT, UPDATE, DELETE Y SELECT).

**Crear una base de datos**

Para crear una base de datos, SQL proporciona la sentencia CREATE DATABASE cuya sintaxis es:

CREATE DATABASE <base de datos>

Esta sentencia especifica el nombre de la base de datos que se desea crear. Cuando desee eliminarla, ejecute la sentencia:

DROP DATABASE <base de datos>

**Crear una tabla**

Para crear un atabla, SQL proporciona la sentencia CREATE TABLE. Esta sentencia especifica el nombre de la tabla, los nombres y tipos de las columnas de la tabla y de las claves primarias y externa de esa tabla (también llamada extranjera, en el sentido de que es importada de otra tabla). Su sintaxis es la siguiente:

CREATE TABLE <tabla>(<columna 1> [, columna 2>] … )

Donde columna n se formula según la sintaxis siguiente:

<columna n> <tipo de dato> [DEFAULT <expresión>]

 [<constante 1> [<constante 2>…]

Algunos de los tipos de datos más utilizados son los siguientes:

|  |
| --- |
| Tipo SQL |
| BOOLEAN |
| INTEGER |
| REAL |
| FLOAT |
| CHAR |
| VARCHAR |
| BINARY |
| DATE |

La cláusula DEFAULT permite especificar un valor por omisión para la columna y, opcionalmente, para indicar la forma o característica de cada columna, se puede utilizar las constantes: NOT NULL (no se permiten valores nulos:NULL), UNIQUE o PRIMARY KEY.

La cláusula PRIMARY KEY se utiliza para definir la columna como clave principal de la tabla. Esto supone que la columna no puede tener valores nulos ni duplicados; es decir, que dos filas no pueden tener el mismo valor en esa columna. Una tabla puede contener una sola restricción PRIMARY KEY.

La cláusula UNIQUE indica que la columna no permite valores duplicados; es decir, que dos filas no pueden tener el mismo valor en esa columna. Una tabla puede contener varias restricciones UNIQUE. Se suele emplear para que el propio sistema compruebe que no se añaden valores que ya existen.

El ejemplo que se muestra a continuación crea la tabla *alumnos*, en la base de datos con la que estemos trabajando, con las columnas id\_alumno, apellidos, noombre, curso y titulación de los tipos especificados. La columna id\_alumno es la clave principal; esto implica que en esa columna todos los valores tienen que ser diferentes y no nulos. El resto de las columnas tampoco permiten valores nulos:

CREATE TABLE alumnos (

 Id\_alumno INTEGER PRIMARY KEY,

 apellidos VARCHAR(24) NOT NULL,

 nombre VARCHAR(18) NOT NULL,

 curso INTEGER NOT NULL,

 titulacion INTEGER NOT NULL

 )

La diferencia entre los tipos CHAR (n) y VARCHAR(n) es que en el primer caso el campo se rellena con espacios hasta n caracteres (longitud fija) y en el segundo no (longitud variable).

**Escribir datos en la tabla**

Para escribir datos en una tabla, SQL proporciona la sentencia INSERT. Esta sentencia agrega una o más filas nuevas a una tabla. Su sitaxis, de forma simplificada, es la siguiente:

INSERT [INTO] <tabla> [(<columna 1>[,<columna 2>] …)]

 VALUES (<expresión 1>[,<expresión 2>]…)….

INSERT [INTO] … SELECT … FROM …

Donde *tabla* es el nombre de la tabla en la que se desean insertar las filas, argumento que va seguido por una lista con los nombres de las columnas que van a recibir los datos especificados por la lista de valores que siguen a la cláusula VALUES. Las columnas no especificadas en la lista reciben el valor NULL, si lo permiten, o el valor predeterminado, si se especificó. Si todas las columnas reciben datos, se puede omitir la lista con los nombres de las columnas.

En el ejemplo que se muestra a continuación añade a la tabla *alumnos* una nueva fila con los valores de las columnas especificados:

INSERT INTO alumnos

 VALUES (324555, ‘Aguirre soriano’, ‘ Leticia’,4,03)

**Modificar datos de una tabla**

Para modificar datos de una tabla, SQL proporciona la sentencia UPDATE. Esta sentencia puede cambiar los valores de filas individuales, grupos de filas o todas las filas de una tabla. Su sintaxis es la siguiente:

UPDATE <tabla>

 SET <columna 1 = (<expresión 1> | NULL)

 [, <columna 2 = (<expresión 2> | NULL)] …

 WHERE <condición de búsqueda>

La cláusula SET contiene una lista separada por comas de las columnas que deben actualizarse y el nuevo valor de cada columna. El valor suministrado por las expresiones incluye elementos tales como constantes, valores seleccionados de una columna de otra tabla, o valores calculados por una expresión compleja. La cláusula FROM identifica la tabla y la clusula WHERE especifica la condición de búsqueda que define la fila de la tabla cuyas columnas se desean modificar.

El ejemplo que se muestra a continuación modifica en la tabla *alumnos* los apellidos del alumno que tiene la clave especificada:

UPDATE alumnos

 SET apellidos = ‘DIAZ DAOIZ’

 WHERE id\_alumno =5142381

**Borrar registros de una tabla**

Para borrar registros en una tabla, SQL proporciona la sentencia DELETE. Esta sentencia quita una o varias filas de una tabla. Una forma simplificada de la sintaxis de DELETE es:

DELETE FROM <tabla> WHERE <condición de búsqueda>

El argumento *tabla* nombra la tabla de la que se van a eliminar las filas. Se eliminan todas las filas que reúnan los requisitos de la condición de búsqueda de la cláusula WHERE. Si no se especifica una cláusula WHERE, se eliminan todas las filas de la tabla.

Cualquier tabla de la que se hayan quitado todas las filas sigue permaneciendo en la base de datos. La instrucción DELETE sólo elimina filas de la tabla; si se quiere quitar la tabla de la base de datos, hay que ejecutar la sentencia:

DROP TABLE <table>

El ejemplo que se muestra a continuación quita de la tabla *alumnos* el alumno que tiene la clave especificada:

DELETE FROM alumnos WHERE id\_alumno =324555

**Seleccionar datos de una tabla**

Para seleccionar datos de una tabla, SQL proporciona la sentencia SELECT. Las cláusulas principales de esta sentencia se pueden resumir del modo siguiente:

SELECT [ALL | DISTINCT] <lista de selección>

 FROM <tablas>

 WHERE <condiciones de selección>

[ORDER BY <columna 1> [ASC | DESC][, <columna 2> [ASC|des]]…]

Las cláusulas de una instrucción SELECT deben especificarse en el orden especificado.

El argumento *lista de selección* describe las columnas del conjunto de resultados. Es una lista de expresiones separadas por comas. Cada expresión de lista de selección suele ser una referencia a una columna de la tabla de la que provienen los datos, aunque puede ser cualquier otra expresión. Al usar la expresión \* en una lista de selección se especifica que se devolverán todas las columnas de la tabla origen.

La cláusula DISTINCT elimina las repeticiones del conjunto de resultados obtenido por SELECT y ALL especifica que pueden aparecer filas duplicadas en el conjunto de resultados; es el valor predeterminado.

La cláusula FROM especifica una lista de las tablas de donde se recuperan los datos del conjunto de resultados.

La cláusula WHERE describe un filtro que define las condiciones que debe cumplir cada fila de las tablas de origen para satisfacer los requisitos de la instrucción SELECT. Sólo las filas que cumplen las condiciones contribuyen con datos al conjunto de resultados. Los datos de las filas que no cumplen las condiciones no se usan.

La cláusula ORDER BY define el orden de las filas del conjunto de resultados y especifica las columnas que intervienen en la clasificación. Las palabras clave ASC y DESC se utilizan para especificar si las filas se ordenan en una secuencia ascendente o descendente. Por omisión se supone ASC.

El ejemplo siguiente lista todas las filas de la tabla alumnos:

SELECT \* FROM alumnos

Este otro ejemplo lista todas las filas de la tabla alumnos ordenadas ascendentemente por los apellidos:

SELECT \* FROM alumnos ORDER BY apellidos

SQL permite utilizar los operadores <,<=,>,>=,<>,AND, OR, NOT, IS NULL, LIKE, BETWEEN, IN ALL, ANY, etc. El ejemplo siguiente lista los alumnos de la tabla *alumnos* que estén en los cursos 1,2 y 3 de la titulación 5:

SELECT \* FROM alumnos WHERE curso <= 3 AND titulacion=5

El siguiente ejemplo lista todos los alumnos de la tabla alumnos que estén matriculados en los cursos 4 y 5 de la titulación 7:

SELECT \* FROM alumnos WHERE curso IN(4,5) AND titulacion=7

## 2.3 MySQL

MySQL surgió como un intento de conectar el gestor mSQL a las tablas propias de MySQL AB, usando sus propias rutinas a bajo nivel. Tras unas primeras pruebas, vieron que mSQL no era lo bastante flexible para lo que necesitaban, por lo que tuvieron que desarrollar nuevas funciones. Esto resultó en una interfaz SQL a su base de datos, con una interfaz totalmente compatible a mSQL.

MySQL empezó como un proyecto de programación hace unos 10 años, cuando un programador sueco decidió crear su propio gestor de datos para la aplicación que estaba desarrollando. Hasta ese momento usaba msql, pero vio que se le quedaba corto en algunos aspectos que el podría mejorar directamente. Pronto MySQL fue el número 1 en rendimiento en el ámbito de las bases de datos de código abierto. Posteriormente se creo una empresa sueca, llamada Tux que se dedicó a crear nuevas versiones de MySQL y comercializarlo en ciertos casos. Dentro de sus características se encuentran:

* Disponibilidad en gran cantidad de plataformas y sistemas.
* Diferentes opciones de almacenamiento según si se desea velocidad en las operaciones o el mayor número de operaciones disponibles.
* Transacciones y claves foráneas.
* Conectividad segura.
* Replicación.
* Búsqueda e indexación de campos de texto.

Existen dos tipos de comandos en MySQL:

* DDL crean y definen nuevas bases de datos, campos e índices.
* DML generan consultas para ordenar, filtrar y extraer datos.

## 2.3.1 Comandos DDL

Con estos comandos se especifican las funciones de definición de datos de ahí su nombre **DDL (Data Definition Language)** e incluye los comandos:

### CREATE

Este comando es utilizado para crear nuevas Bases de Datos o nuevas tablas dentro de una Base de Datos específica.

Sintaxis:

DROP DATABASE IF EXISTS nombre\_BD;

 CREATE DATABASE nombre\_BD;

Un ejemplo de aplicación de este comando para crear una nueva Base de Datos es el siguiente:

DROP DATABASE IF EXISTS videoteca;

CREATE DATABASE videoteca

En el caso de la creación de tablas la sintaxis es la siguiente:

USE nombre\_BD;

DROP TABLE IF EXISTS nombre\_tabla;

CREATE TABLE nombre\_tabla(

 var1 tipo1 [NOT NULL] [PRIMARY KEY],

 … varN tipoN

);

USE videoteca;

DROP TABLE IF EXISTS pelicula;

CREATE TABLE pelicula(

 titulo VARCHAR(64),

 director VARCHAR(128),

 actor VARCHAR(128)

);

USE videoteca;

DROP TABLE IF EXISTS rentas;

CREATE TABLE rentas(

 id\_cliente INT NOT NULL PRIMARY KEY,

 cliente VARCHAR(64),

 titulo VARCHAR(128)

);

En la creación de tablas es muy importante la generación de llaves foráneas. Los comandos para establecer llaves foráneas incluyen:

* Foreign Key es utilizado para designar llaves foráneas en una tabla.
* References establece cuál es la tabla de donde se obtiene la llave foránea.
* On delete cascade indica que al eliminar registros en la otra tabla se eliminan de la tabla actual
* On update cascade indica que al actualizar registros en la otra tabla se actualizan en la tabla actual

Sintaxis:

USE nombre\_BD;

DROP TABLE IF EXISTS nombre\_tabla;

CREATE TABLE nombre\_tabla(

 var1 tipo1 [NOT NULL] [PRIMARY KEY],

 var\_foranea tipo x [NOT NULL] …

 varN tipoN

FOREIGN KEY(var\_foranea)

REFERENCES otra\_Tabla(var\_foranea) ON DELETE CASCADE ON UPDATE CASCADE

);

USE videoteca;

DROP TABLE IF EXISTS INVENTARIO;

CREATE TABLE INVENTARIO(

 Id\_inventario integer PRIMARY KEY

 precio integer,

 id\_renta integer

FOREIGN KEY(id\_renta)

REFERENCES rentas(id\_renta) ON DELETE CASCADE ON UPDATE CASCADE

);

### DROP

Este comando es empleado para eliminar Bases de Datos y tablas.

Sintaxis

DROP Database nombre\_BD;

DROP table nombre\_tabla;

Ejemplo

DROP table rentas;

### ALTER

Este comando es utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Sintaxis:

 ALTER TABLE nombre\_tabla

 ADD COLUMN campo1 tipo\_campo1;

 ALTER TABLE nombre\_tabla

 DROP COLUMN campo1;

##

ALTER TABLE pelicula

ADD COLUMN categoria Varchar(20) ;

ALTER TABLE pelicula DROP COLUMN categoria;

## 2.4 Comandos DML

Este tipo de comandos se encargan de la manipulación de los datos es por eso su nombre **DML(Data Manipulation Language)**, incluye los comandos:

### INSERT

Este comando es utilizado para cargar lotes de datos en la base de datos en una única operación.

Sintaxis:

 Use nombre\_BD;

 INSERT INTO Tabla (campo1,

 campo2, .., campoN)

 VALUES (valor1, valor2, ..., valorN);

INSERT INTO pelicula (titulo, director, actor) VALUES( 'Blade Runner', 'Ridley Scott', 'Harrison Ford' );

INSERT INTO pelicula (titulo, director, actor) VALUES( 'Alien', 'Ridley Scott', 'Sigourney Weaver' );

### SELECT

Este comando es utilizado para consultar registros de la base de datos que satisfagan un criterio determinado

Sintaxis:

 Use nombre\_BD;

 SELECT [DISTINCT] <lista\_columnas> | \*

FROM <lista\_tablas>

[WHERE <predicado>]

Ejemplo

 SELECT \* FROM pelicula;

 SELECT \* FROM pelicula WHERE director='Ridley Scott';

 SELECT titulo FROM pelicula WHERE director='Ridley Scott‘

ORDER BY titulo;

### UPDATE

Utilizado para modificar los valores de los campos y registros especificados

Sintaxis:

 Use nombre\_BD;

 UPDATE Tabla SET

 Campo1=Valor1,

 Campo2=Valor2, ... CampoN=ValorN

 WHERE Criterio;

USE videoteca;

UPDATE pelicula

SET titulo='Star Wars'

WHERE titulo='La guerra de las galaxias';

DELETE

### DELETE

Utilizado para eliminar registros de una tabla de una base de datos.

Sintaxis:

DELETE \* FROM Tabla WHERE criterio;

DELETE \* FROM pelicula WHERE titulo='Star Wars';

## 2.5 Conexión de MySQL con PHP

La interacción de PHP con MySQL es muy sencilla, el único requerimiento es que el usuario de PHP conozca los permisos y contraseñas necesarios para acceder a la Base de Datos que solicite.

Un ejemplo de consulta es el siguiente:

<?php

$link = mysql\_connect('localhost', 'root', 'miclave');

if (!$link) {

 die('Could not connect: ' . mysql\_error());

}

else

{echo "Conexión exitosa<br>\n";}

$db\_selected = mysql\_select\_db('videoteca', $link);

if (!$db\_selected) {

 die ('Acceso Denegado : ' . mysql\_error());

}

else

{

echo "Ejecución de consulta aceptada<br>\n";

}

mysql\_close($link);

?>

Para mostrar los resultados de la consulta realizamos lo siguiente:

<?php

//bloque de conexión

$link = mysql\_connect('localhost', 'root', 'miclave');

if (!$link) {

 die('Could not connect: ' . mysql\_error());}

else

{echo "Conexión exitosa<br>\n";}

//bloque de ejecución de consulta

$db\_selected = mysql\_select\_db('videoteca', $link);

if (!$db\_selected) {

 die ('Acceso Denegado : ' . mysql\_error());

}

else

{ echo "Ejecución de consulta aceptada<br>\n";

}

$sql = " SELECT \* from pelicula ";

$result = mysql\_query($sql) ;

if (!$result) { echo "Error en la consulta"; exit(); }

while($data=mysql\_fetch\_object($result))

{

 echo "El Título es:". $data->titulo."<br>\n";

 echo "\n El director es:". $data->director."<br>\n";

 echo "\n El actor es:". $data->actor."<br>\n";

}

mysql\_free\_result ($result);

mysql\_close($link);

?>

#

# 3. Lenguaje del lado del servidor: PHP

PHP es el heredero de un producto anterior, llamado PHP/FI. PHP/FI fue creado por Rasmus Lerdorf en 1995, inicialmente como un simple conjunto de scripts de Perl para controlar los accesos a su trabajo online. Llamó a ese conjunto de scripts 'Personal Home Page Tools'. Según se requería más funcionalidad, Rasmus fue escribiendo una implementación en C, capaz de comunicarse con bases de datos, y permitía a los usuarios desarrollar sencillas aplicaciones Web dinámicas. Rasmus eligió liberar el código fuente de PHP/FI para que cualquiera pudiese utilizarlo, así como arreglar errores y mejorar el código.

PHP permite embeber pequeños fragmentos de código dentro de la página HTML y realizar determinadas acciones de una forma fácil y eficaz sin tener que generar programas programados íntegramente en un lenguaje distinto al HTML. Por otra parte, y es aquí donde reside su mayor interés con respecto a los lenguajes pensados para los CGI, PHP ofrece un sinfín de funciones para la explotación de bases de datos de una manera llana, sin complicaciones.

Como funcionalidades primordiales en PHP se consideran:

* Funciones de correo electrónico: envío de correos individual o por grupos, parametrizando toda una serie de aspectos tales como el e-mail de procedencia, asunto, destinatario.
* Gestión de bases de datos : El lenguaje PHP ofrece interfaces para el acceso a la mayoría de las bases de datos comerciales y por ODBC a todas las bases de datos posibles en sistemas Microsoft.
* Gestión de archivos: mediante operaciones de creación, borrado, modificación, además de ofrecer transferencia de archivos.
* Tratamiento de imágenes: mediante funciones de automatización de formato, envío de lotes de imágenes y funciones de graficado.

## 3.1 Entorno de desarrollo de PHP

El primer paso es descargar los programas necesarios como los siguientes:

* Apache: [www.apache.org](http://www.apache.org) apache\_1\_3\_x\_win32.exe
* MySQL: [www.mysql.com](http://www.mysql.org) mysql-shareware-3.22.34-win.zip
* PHP: [www.php.net](http://www.php.net) php-3.0.x-win32.zip

Para instalar Apache con PHP hay que hacer modificaciones a archivos de configuración, si se necesita MySQL hace falta habilitar un módulo en PHP. Así para facilitarnos la instalación de un servidor existen algunos paquetes que nos instalan todo el software y sin necesidad de configurarlo como el caso de WAMP (<http://www.wampserver.com/en/index.php>).

WAMP es un paquete sólo para Windows que instala Apache, PHP 5, MySQL 5, PHPmyadmin y SQLitemanager de un sólo paso. Cuenta con varias utilerías como un intérprete Perl y el Zend Optimizer, y para mayor compatibilidad también es posible instalar PHP 4 y Apache 2.

Para ejecutar cualquier script en la plataforma WAMP, bastará con guardar los documentos con la extensión PHP en el directorio WAMP/www.

## 3.2 Sintaxis Básica

PHP se escribe dentro de la propia página web, junto con el código HTML y, como para cualquier otro tipo de lenguaje incluido en un código HTML, en PHP necesitamos especificar cuáles son las partes constitutivas del código escritas en este lenguaje. Esto se hace, como en otros casos, delimitando nuestro código por etiquetas. Se pueden utilizar distintos modelos de etiquetas en función de nuestras preferencias y costumbres. Estos modos de abrir y cerrar las etiquetas son:

<? y ?>

<% y %>

<?php y ?>

<script languaje="php">

El modo de funcionamiento de una página PHP, no difiere del clásico para una página dinámica de lado servidor: El servidor va a reconocer la extensión correspondiente a la página PHP (phtml, php, php4,...) y antes de enviarla al navegador va a encargarse de interpretar y ejecutar todo aquello que se encuentre entre las etiquetas correspondientes al lenguaje PHP. El resto, lo enviara sin más ya que, asumirá que se trata de código HTML absolutamente comprensible por el navegador.

Otra característica general de los scripts en PHP es la forma de separar las distintas instrucciones. Para hacerlo, hay que acabar cada instrucción con un punto y coma ";".

Además otro punto esencial para la documentación de cualquier programa son los comentarios, que son frases o palabras que se incluyen en el código para comprenderlo más fácilmente al volverlo a leer un tiempo después y que, por supuesto, la computadora ignora.

Con estos preliminares, se puede construir el script básico como el que aparece a continuación:

<html>

<body>

<?php

$myvar = "Hola. \n";

//Esto es un comentario

echo “es mi primer script en PHP \n";

//Esto es un comentario

/\*En este caso

mi comentario ocupa

varias lineas, lo ves? \*/

echo $myvar;

?>

</body>

</html>

Una vez escrito esto lo guardamos en un fichero con la extensión php, y lo colocamos en nuestro servidor de Wampserver, http://localhost/script1.php. Ahora en el navegador se observará una línea con el texto "Hola. Este es mi primer script en PHP".

Lo primero que apreciamos en el script son sus delimitadores. En la primera línea del script vemos **<?php** que nos indica que comienza un script en PHP, y en la última colocamos **?>** para indicar el final del script. Hay que destacar que todas las líneas que se encuentre entre estos delimitadores deben acabar en punto y coma, excepto las sentencias de control (if, switch, while, etc.).

Para que el servidor envié texto al explorador utilizaremos la instrucción **echo**, aunque también podemos utilizar **printf**.

Finalmente, se observa que la palabra myvar comienza con el signo dólar ($). Este símbolo le indica a PHP que es una variable. También se observa en el texto que le asignamos a la variable termina con \n, esto no se imprime sólo sirve para indicarle al navegador una nueva línea.

## 3.3 Variables y Operadores

Todas la variables son precedidas por el signo dólar ($), y se le asigna un contenido con el signo igual (=). Con las variables, PHP distingue entre mayúsculas y minúsculas.

<html>

<body>

<?php

$myvar = "SEVILLA \n";

$Myvar = "MADRID \n";

//Esto imprimirá SEVILLA

echo $myvar;

//Esto imprimirá MADRID

ECHO $Myvar;

?>

</body>

</html>

El uso de la barra invertida, como en \n, no es obligatorio, pero ayuda a la depuración del código que enviamos al navegador, además del \n existen otros usos:

### 3.3.1 Operadores Aritméticos:

$a + $b Suma o $a - $b Resta

$a \* $b Multiplicación

$a / $b División

$a % $b Resto de la división de $a por $b

$a++ Incrementa en 1 a $a

$a-- Resta 1 a $a

### 3.3.2 Operadores de Cadenas:

El único operador de cadenas que existe es el de concatenación, el punto.

$a = "Hola";

$b = $a . "Mundo"; // Ahora $b contiene "Hola Mundo"

En este punto hay que hacer una distinción, la interpretación que hace PHP de las simples y dobles comillas. En el segundo caso PHP interpretará el contenido de la cadena.

$a = "Mundo";

echo = 'Hola $a'; //Esto escribirá "Hola $a"

echo = "Hola $a"; //Esto escribirá "Hola Mundo"

### 3.3.3 Operadores de Comparación:

$a < $b $a menor que $b

$a > $b $a mayor que $b

$a <= $b $a menor o igual que $b

$a >= $b $a mayor o igual que $b

$a == $b $a igual que $b

$a != $b $a distinto que $b

### 3.3.4 Operadores Lógicos:

$a AND $b Verdadero si ambos son verdadero

$a && $b Verdadero si ambos son verdadero

$a OR $b Verdadero si alguno de los dos es verdadero

$a !! $b Verdadero si alguno de los dos es verdadero

$a XOR $b Verdadero si sólo uno de los dos es verdadero !$a Verdadero si $a es falso, y recíprocamente.

### 3.3.5 Operadores de Asignación:

$a = $b Asigna a $a el contenido de $b

$a += $b Le suma a $b a $a

$a -= $b Le resta a $b a $a

$a \*= $b Multiplica $a por $b y lo asigna a $a

$a /= $b Divide $a por $b y lo asigna a $a

$a .= $b Añade la cadena $b a la cadena $a

## 3.4 Sentencias de Control

Las sentencias de control permiten ejecutar bloque de códigos dependiendo de ciertas condiciones. Para PHP el 0 es equivalente a Falso y cualquier otro número es Verdadero.

### 3.4.1 IF...ELSE

La sentencia **IF...ELSE** permite ejecutar un bloque de instrucciones si la condición es Verdadera y otro bloque de instrucciones si ésta es Falsa. Es importante tener en cuenta que la condición que evaluemos ha de estar encerrada entre paréntesis (esto es aplicable a todas la sentencias de control).

**if (condición) {**

**Este bloque se ejecuta si la condición es VERDADERA**

**} else {**

**Este boque se ejecuta si la condición es FALSA**

**}**

Existe una forma sencilla de usar la sentencia **IF** cuando no tenemos que usar **ELSE** y solo tenenemos que ejecutar una línea de código.

if ($a > 4) echo "$a es mayor que 4";

### 3.4.2 IF...ELSEIF...ELSE

La sentencia **IF...ELSEIF...ELSE** permite ejecuta varias condiciones en cascada. Para este caso veremos un ejemplo, en el que utilizaremos los operadores lógicos.

<?php

if ($nombre == ""){

 echo "Tú no tienes nombre";

} elseif (($nombre=="eva") OR ($nombre=="Eva")) {

 echo "Tu nombre es EVA";<

} else {

 echo "Tu nombre es " . $nombre;

}

?>

### 3.4.3 WHILE

La sentencia **WHILE** ejecuta un bloque de código mientras se cumpla una determinada condición.

<?php

$num = 1;

while ($num < 5) {

 echo $num;

 $num++

}

?>

### 3.4.4 DO...WHILE

Esta sentencia es similar a **WHILE**, salvo que con esta sentencia primero ejecutamos el bloque de código y después se evalúa la condición, por lo que el bloque de código se ejecuta siempre al menos una vez.

<?php

$num = 1;

do {

 echo $num;

 if ($num == 3){

 echo "Aquí nos salimos \n";

 break

}

$num++

} while ($num < 5);

?>

### 3.4.5 FOR

El bucle **FOR** resulta muy útil cuando debemos ejecutar un bloque de código a condición de que una variable se encuentre entre un valor mínimo y otro máximo.

<?php

for ($num = 1; Snum <=5; $num++){

 echo $num;

 if ($num == 3){

 echo "Aquí nos salimos \n";

 break

 }

 } ?>

## 3.5 Tablas

Las tablas son utilizadas en las funciones relacionadas a las bases de datos,  lo hacen en forma de tabla. En PHP disponemos de dos tipos de tablas. El primero sería el clásico, utilizando índices:

<?php

$ciudad[] = "París";

$ciudad[] = "París";

$ciudad[] = "Roma";

$ciudad[] = "Sevilla";

$ciudad[] = "Londres";

print ("yo vivo en " . $ciudad[2] . "<BR>\n");

?>

Esta es una forma de asignar elementos a una tabla, pero una forma más formal es utilizando la función **array**

<?php

$ciudad = array("París", "Roma", "Sevilla", "Londres");

//contamos el número de elementos de la tabla

$numelentos = count($ciudad);

//imprimimos todos los elementos de la tabla

for ($i=0; $i < $numelentos; $i++)

{

print ("La ciudad $i es $ciudad[$i] <BR>\n");

}

?>

Un segundo tipo, son las **tablas asociativas**, en las cuáles a cada elemento se le asigna un valor (key) para acceder a él.

Para entenderlo, supongamos que tenemos una tabla en la que cada elemento almacena el número de visitas a una página web por cada día de la semana.

Utilizando el método clásico de índices, cada día de la semana se representaría por un entero, 0 para lunes, 1 para martes, etc.

$visitas[0] = 200;

$visitas[1] = 186;

Con las tablas asociativas sería:

$visitas["lunes"] = 200;

$visitas["martes"] = 186;

La manipulación de las tablas asociativas se a hace través de funciones que actúan sobre un puntero interno que indica la posición. Por defecto, el puntero se situa en el primer elemento añadido en la tabla, hasta que es movido por una función:

**current** - devuelve el valor del elemento que indica el puntero
**pos**      - realiza la misma función que **current**
**reset**   - mueve el puntero al **primer** elemento de la tabla
**end**     - mueve el puntero al **último** elemento de la tabla
**next**    - mueve el puntero al elemento **siquiente**
**prev**    - mueve el puntero al elemento **anterior**
**count** - devuelve el número de  elementos de una tabla.

 Veamos un ejemplo de las funciones anteriores:

<?php

$semana = array("lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domindo");

echo count($semana); //7

//situamos el puntero en el primer elemento

reset($semana);

echo current($semana); //lunes

next($semana);

echo pos($semana); //martes

end($semana)

echo pos($semana); //domingo

prev($semana);

echo current($semana); //sábado

?>

Recorrer una tabla con las funciones anteriores se hace un poco tedioso, para ello se recomienda utilizar la función **each().**

<?php

$visitas = array("lunes"=>200, "martes"=>186, "miércoles"=>190, "jueves"=>175);

reset($visitas);

while (list($clave, $valor) = each($visitas))

{

echo "el día $clave ha tenido $valor visitas<BR>";

}

?>

La función **each()**devuelve el valor del elemento actual, en este caso, el valor del elemento actual y su clave, y desplaza el puntero al siguiente, cuando llega al final devuelve **FALSO**, y termina el bucle **while()**.

# 3.6 Funciones

Una Función no es más que un bloque de código con una serie de parámetros y devuelve un valor. Para declarar una funcion debemos utilizar la instrucción **function** seguido del nombre que le vamos a dar, y después entre parentesis la lista de argumentos separados por comas.

**function nombre\_de\_funcion (arg\_1, arg\_2, ..., arg\_n)**

**{**

**bloque de código**

**}**

Mediante el uso de la instrución **return** terminamos la ejecución del código de una función y devolvemos un valor. Podemos tener varios **return** en nuestra función, pero por lo general, cuantos más **return** tengamos menos reutilizable será nuestra función, por ejemplo.

<?php

function mayor ($x, $y)

{

if ($x > $y) {

 return $x." es mayor que".$y;

} else {

return $y." es mayor que".$x;

 }

}

?>

## 3.6.1 Llamado de scripts

Por lo general, todos los script tienen partes de código iguales, las funciónes **include()** y **require()** nos van ahorrar muchas de estas lineas de código. Ambas funciones hacen una llamada a un determinado archivo pero de dos maneras diferentes, con **include()**, insertamos lo que contenga el fichero que llamemos de manera literal en nuestro script, mientras que con **require()**, le decimos que el script necesitará parte de código de se encuentra en el fichero que llama **require()**. Veamos el siguiente ejemplo.

<?php

include ("header.inc");

echo "Hola Mundo";

include ("footer.inc");

?>

Si tenemos encuenta que el archivo **header.inc** contiene: <html> <body> y el archivo **footer.inc** contiene: </body> </html> lo que ahorra escribir continuamente líneas de código empleadas en la mayoría de las aplicaciones.

Ahora veamos el script de ejemplo para la función **require()**:

<?php

require ("config.inc");

include ("header.inc");

echo $cadena;

include ("footer.inc");

?>

Donde el archivo **config.inc** contiene: <?php $cadena = "Hola Mundo"; ?>

**Tiempo y fecha**

En este apartado veremos ejemplos prácticos de funciones relacionadas con el tiempo y la fecha.

La función time devuelve el número de segundos transcurridos desde el 1 de Enero de 1970. A esta forma de expresar fecha y hora se le denomina **tmestamp**.

La función **date** devuelve una cadena formateada según los código de formato. Si no le pasamos la variable **timestamp** nos devuelve la cadena formateada para la fecha y la hora actual.

Los códigos de formato para la función **date** son:

|  |  |
| --- | --- |
| CODIGO  |  DESCRIPCIÓN  |
| a  |  am o pm  |
| A  |  AM o PM  |
| d  |  Día del mes con ceros  |
| D  |  Abreviatura del día de la semana (inglés)  |
| F  |  Nombre del mes (inglés)  |
| h  |  Hora en formato 1-12  |
| H  |  Hora en formato 0-23  |
| i  |  Minutos  |
| j  |  Día del mes sin ceros  |
| l  |  Dia de la semana  |
| m  |  Número de mes (1-12)  |
| M  |  Abreviatura del mes (inglés)  |
| s  |  Segundos  |
| y  |  Año con 2 dígitos  |
| Y  |  Año con 4 dígitos  |
| z  |  Dia del año (1-365)  |

Para ver algunos ejemplos supongamos que ahora es el 7 de abril de 2000 a las 14 horas 30 minutos y 22 segundos:

date("d-m-Y") -> 07-04-2000

date("H:i:s") -> 14:30:22

date("YmdHis") -> 20000407143022

date(d-m-Y H:i, time()) -> el momento actual

La funcion **mktime** devuelve una variable de tipo **timestamp** a partir de las coordenadas dadas. La principal utilidad de esta funcion es la de añadir o quitar una determinada cantidad de fecha u horas a una dada.

<?PHP

function restarDias($numdias, $date) {

 if (isset($date)) {

 $date = time();

 }

list($hora, $min, $seg, $dia, $mes, $anno) = explodest($hora, $min,

 $seg, $dia, $mes, $anno) = explode( " ", date( "H i s d m Y"));

$d = $dia - $numdias;

$fecha = date("d-m-Y", mktime($hora, $min, $seg, $mes, $d, $anno));

return $fecha;

}

echo restarDias(5)."<BR>";

echo restarDias(10)."<BR>";

?>

La función **checkdate** comprueba si una fecha es válida, si es así devuelve TRUE y si no, es FALSE. Una fecha se considera válida si el año está entre 1900 y 32767, el mes entre 1 y 12, y el dia es menor o igual que número de días total del mes en cuestión.

<?PHP

if (checkdate(31, 2, 2000)) {

 echo "La fecha es correcta";

} else {

 echo "La fecha es incorrecta";

}

?>

Para el ejemplo anterior nos daría que la fecha es incorrecta, febrero nunca tiene un día 31.

## 3.7 Clases en PHP

PHP no es un lenguaje orientado a objetos, pero implementa las características que permiten definir las clases. Por ejemplo, la clase automóvil, la cual tendrá diversas variables, $color, $modelo, $marca, $potencia, $matricula y habrá una serie de funciones que actuarán sobre ésta como Precio(), Acelerar(), Frenar(), Girar()y Reparar().

Como ejemplo se tiene la clase mysql, que servirá para realizar consultas a las bases de datos MySQL.

<?php

class DB\_mysql {

/\* variables de conexión \*/

var $BaseDatos;

var $Servidor;

var $Usuario;

var $Clave;

/\* identificador de conexión y consulta \*/

var $Conexion\_ID = 0;

var $Consulta\_ID = 0;

/\* número de error y texto error \*/

var $Errno = 0;

var $Error = "";

/\* Método Constructor: \*/

function DB\_mysql($bd = "", $host = "localhost", $user = "nobody", $pass = "") {

 $this->BaseDatos = $bd;

 $this->Servidor = $host;

 $this->Usuario = $user;

 $this->Clave = $pass;

}

/\*Conexión a la base de datos\*/

function conectar($bd, $host, $user, $pass){

 if ($bd != "") $this->BaseDatos = $bd;

 if ($host != "") $this->Servidor = $host;

 if ($user != "") $this->Usuario = $user;

 if ($pass != "") $this->Clave = $pass;

 // Conectamos al servidor

 $this->Conexion\_ID =

 mysql\_connect($this->Servidor, $this->Usuario, $this->Clave);

 if (!$this->Conexion\_ID) {

 $this->Error = "Ha fallado la conexión.";

 return 0;

 }

 //seleccionamos la base de datos

 if (!@mysql\_select\_db($this->BaseDatos, $this->Conexion\_ID)) {

 $this->Error = "Imposible abrir ".$this->BaseDatos ;

 return 0;

 }

 /\* Si hemos tenido éxito conectando devuelve

 el identificador de la conexión, sino devuelve 0 \*/

 return $this->Conexion\_ID;

}

/\* Ejecuta un consulta \*/

function consulta($sql = ""){

 if ($sql == "") {

 $this->Error = "No ha especificado una consulta SQL";

 return 0; }

 //ejecutamos la consulta

 $this->Consulta\_ID = @mysql\_query($sql, $this->Conexion\_ID);

 if (!$this->Consulta\_ID) {

 $this->Errno = mysql\_errno();

 $this->Error = mysql\_error();

 /\* Si hemos tenido éxito en la consulta devuelve

 el identificador de la conexión, sino devuelve 0 \*/

 return $this->Consulta\_ID; }

/\* Devuelve el número de campos de una consulta \*/

function numcampos() {

 return mysql\_num\_fields($this->Consulta\_ID); }

/\* Devuelve el número de registros de una consulta \*/

function numregistros(){

 return mysql\_num\_rows($this->Consulta\_ID); }

 /\* Devuelve el nombre de un campo de una consulta \*/

function nombrecampo($numcampo) {

 return mysql\_field\_name($this->Consulta\_ID, $numcampo); }

/\* Muestra los datos de una consulta \*/

function verconsulta() {

 echo "<table border=1>\n";

 // mostramos los nombres de los campos

 for ($i = 0; $i < $this->numcampos(); $i++){

 echo "<td><b>".$this->nombrecampo($i)."</b></td>\n";

 }

 echo "</tr>\n";

 // mostrarmos los registros

 while ($row = mysql\_fetch\_row($this->Consulta\_ID)) {

 echo "<tr> \n";

 for ($i = 0; $i < $this->numcampos(); $i++){

 echo "<td>".$row[$i]."</td>\n";

 }

 echo "</tr>\n";

 } //fin de while

 } // fin de verconsulta

 } //fin de la Clase DB\_mysql

?>

Para crear una clase utilizamos la sentencia **class**, y además hemos creado una función con el mismo nombre que la clase, a esa función se le llama constructor y se ejecutará cada vez que definamos una variable de esa clase. Otra cosa importante en las clases es el operador **->**, con el que indicamos una variable o método (parte derecha del operador) de una clase (parte izquierda del operador). Para hacer referencia a la clase que estamos creando dentro de su definición, debemos utilizar el operador **this**.

Y ahora veamos un ejemplo de la clase que hemos creado, y supongamos que el código anterior lo hemos guardado en un fichero llamado clase\_mysql.inc.php.

<body> <html>

<?php

require ("clase\_mysql.inc.php");

$miconexion = new DB\_mysql ;

$miconexion->conectar("mydb", "localhost", "nobody", "");

$miconexion->consulta("SELECT \* FROM agenda");

$miconexion->verconsulta();

?>

</body> </html>

## Envío de Mensajes

Mediante PHP se puede enviar información y variables a la misma página o a diferentes, un ejemplo básico es el siguiente:

<HTML>

<HEAD>

<TITLE>origen.html</TITLE>

</HEAD>

<BODY>

<a href="destino.php?saludo=hola&texto=Esto es una variable texto">Paso variables saludo y texto a la página destino.php</a>

</BODY>

</HTML>

Esta es la página donde se origina el mensaje

Esta es la página donde se recibe el mensaje

<HTML>

<HEAD>

<TITLE>destino.php</TITLE>

</HEAD>

<BODY>

<?php

echo "Variable \$saludo: $saludo <br>\n";

echo "Variable \$texto: $texto <br>\n"

?>

</BODY>

</HTML>

# 3.8 Formularios HTML

Seguramente ya ha aprendido cómo acceder a la Web; si ya lo ha hecho, habrá visto muchas páginas con magníficos gráficos, listas, formularios y otros elementos muy atractivos, de igual forma seguramente sabe que el lenguaje utilizado aquí es HTML.

**Que es HTML**

HTML es un lenguaje utilizado para desarrollar páginas y documentos Web. A diferencia de los lenguajes convencionales, HTML utiliza una serie de etiquetas especiales intercaladas en un documento de texto sin formato. Dichas etiquetas serán posteriormente interpretadas por los exploradores encargados de visualizar la página o el documento Web con el fin de establecer el formato.

## Creando Formularios

Los formularios permiten crear interfaces gráficas de usuario en el seno de una página Web. Los componentes de estas interfaces, denominados también controles, serán cajas de texto, cajas para clave de acceso, botones de pulsación, botones de opción, casillas de verificación, menús, tablas, listas desplegables, etc.

Para crear un formulario, se utiliza la etiqueta **form**. Por ejemplo, en el segmento de código siguiente muestra cómo es el esqueleto de un formulario HTML:

< form method = {get | post } action=”Fichero para procesar los datos “>

 *Controles que forman el formulario*

 /form>

El atributo **method** es opcional y su valor por omisión es **get** indicando así al navegador que debe agregar los nombres de los campos del formulario y sus datos al URL especificado por **action** (acción a tomar cuando se pulse el botón *enviar*). La cantidad de datos que se pueden concatenar al URL está limitada, truncándose la información en exceso. Esto no ocurre si el valor para este atributo es **post**, en este caso, se transmite un fichero con los datos del formulario que será recibido en el servidor en la entrada estándar del componente de procesamiento.

Los controles que forma parte del formulario se definen dentro de la etiqueta **form**. En general, por cada control se envía al servidor su nombre y su contenido o valor especificado (si no se especifica un valor, se envía uno por omisión).

## Entrada básica de datos

Existe una amplia variedad de controles de entrada de datos. Para crearlos, se utiliza la etiqueta imput y los atributos type y name:

< input

 type ={text | password | checkbox | radio | hidden | image | submit | reset }

 name = “Variable que toma el valor

>

El valor del atributo **type** especifica el tipo de control que se creará, y el del atributo **name** el nombre de la variable que almacenará el dato.

**Caja de texto**

Una caja de texto es un control de entrada de tipo *text*. Se utiliza para solicitar a un usuario que introduzca un dato. Por ejemplo, la siguiente línea de código muestra una caja de texto para solicitar un nombre. El tamaño de la caja es 35 y el valor será almacenado en la variable *nombre*:

Nombre: <input type=”text” name =”nombre” size=”35”>

El valor del atributo **size** especifica el tamaño de la caja. Otros atributos que se pueden utilizar son **value** para especificar un valor inicial, **readonly** (que no lleva asociado ningún valor) para indicar que la caja es de sólo lectura y **max-length** para especificar el número máximo de caracteres que un usuario puede escribir.

**Caja de clave de acceso**

Una caja de clave de acceso es un control de entrada de tipo *password.* Se trata de una caja de texto en la que los caracteres escritos son reemplazados por asteriscos. Por ejemplo, la siguiente línea de código muestra una caja e texto para solicitar clave de acceso. El tamaño de caja es 25, el valor será almacenado en la variable *clave* y el número máximo de caracteres que admite es 20.

Clave de acceso: <input type=”password” name =”clave”

 size=”25” maxlength=”20”>

**Casilla de verificación**

Una casilla de verificación es un control de entrada de tipo *checkbox*. Se trata de un botón que puede presentar dos estados: seleccionado y no seleccionado. Se utilizan para mostrar y registrar opciones que un usuario puede elegir; puede seleccionar varias de un grupo de ellas. Por ejemplo, las siguientes líneas de código muestran tres casillas de verificación. La primera se mostrará seleccionada:

<input type=”checkbox” name =”cv1” value=”1” checked > Opcion 1 <br>

<input type=”checkbox” name =”cv2” value=”2” > Opcion 2 <br>

<input type=”checkbox” name =”cv2” value=”3” > Opcion 3 <br>

Se debe especificar el atributo **name**. El atributo **checked** permite iniciar el estado de una casilla ha seleccionado. Cuando se envíen los datos del formulario, se enviarán el nombre de la variable y el valor que indique su estado.

**Botón de opción**

Un botón de opción es un control de entrada de tipo radio. Igual que ocurre con la casilla de verificación, puede presentar dos estados: seleccionado y no seleccionado. Se utilizan para mostrar y registrar una opción que un usuario puede legir entre varias; cuando selecciona una, la que estaba seleccionada dejará de estarlo. Por ejemplo, las siguientes líneas de código muestran tres botones de opción. El segundo se mostrará seleccionado:

<input type=”radio” name =”opcion” value=”1” > Opcion 1 <br>

<input type=”radio” name =”opcion” value=”2” checked > Opcion 2 <br>

<input type=”radio” name =”opcion” value=”3” > Opcion 3 <br>

Para el comportamiento descrito, todos los botones de opción tendrán el mismo atributo **name** y con un valor distinto del atributo **value**. El valor enviado será el correspondiente al botón seleccionado. El atributo **checked** permitirá seleccionar por omisión uno de los botones de un grupo.

**Parámetros ocultos**

Un parámetro oculto es un control de entrada de tipo *hidden*. En este caso no se muestra ningún campo de entrada de datos al usuario, pero el par variable valor especificado es enviado junto con el formulario.

<input type=”hidden” name =”variable” value=”valor”>

Se suelen utilizar para mantener datos durante una sesión.

**Enviar datos**

Un botón *enviar* es un control de entrada de tipo *submit*. Se utiliza para enviar los datos del formulario, pasando el control al programa indicado por el atributo **action** del formulario. Todo formulario debe tener un botón **submit**, a menos que incluya una caja de texto

<input type=”submit” value=”Enviar datos”>

El atributo **value** específica una etiqueta no editable que se mostrará como título del botón. Lo normal es que este control no envíe datos, pero si se incluye el atributo **name** con un nombre de variable, será enviada la variable con el valor de **value**. Esto puede ser útil para distinguir cuál fue el botón pulsado cuando se incluyan varios.

<input type=”submit” name =”enviar” value=”Enviar”>

<input type=”submit” name =”buscar” value=”Buscar”>

**Borrar los datos de un formulario**

<input type=”reset” value=”Borrar datos”>

**Imágenes**

Una imagen es un control de entrada de tipo *image*. Su finalidad es análoga al botón **submit**, pero en este caso se presenta una imagen en lugar de un botón. Los datos del formulario se enviarán al hacer clic sobre la imagen .jpg o .gif.

<input type=”image” src=”fichero.jpg”>

**Orden de tabulación**

Pata trasladarse de un control a otro en el formulario, el usuario puede utilizar la tecla *tab*. En este caso, el orden en el que los controles serán recorridos queda determinado por su orden de aparición en el formulario, o bien por el atributo **tabindex**. Por ejemplo:

<input type=”text” name=”nombre” size=”30” tabindex=”1”>

El valor de tabindex es un número(1,2,3,…) que corresponde al orden en el que se seleccionará un control cuando el usuario pulse la tecla *tab*. El control que se seleccionará en primer lugar es el 1.

**Caja de texto multilínea**

En ocasiones es necesario permitir al usuario escribir varias líneas de texto libre; por ejemplo, un mensaje. Para esto se utiliza un control denominado *área de texto* (**textarea**) que incluye una barra de desplazamiento vertical:

Mensaje: <br> <textarea name = “mensaje”

 Rows=”5” cols = “20” wrap >

 </textarea>

El valor del atributo **rows** especifica el numero de filas que visualizará el área de texto a la vez y **cols** el número de caracteres por fila. El atributo **wrap**, que no lleva asociado ningún valor, indica que se saltará automáticamente a la línea siguiente cuando se complete la línea en la que se escribe.

**Listas desplegables**

Una lista desplegable permite seleccionar una lista entre varias. Es la mejor alternativa para añadir menús a una interfaz gráfica. La etiqueta que permite crear un control de este tipo es **select**. Las opciones de la lista se especifican utilizando la etiqueta **option**. Por ejemplo, el siguiente código mostrará una lista desplegable con tres opciones, de las cuales aparecerá seleccionada inicialmente la segunda:

<select name = “opcion” >

 <option value = “1” > Opción 1

 <option selected value=”2” > Opción 2

 <option value=”3” > Opción 3

</select>

El atributo **value** de **option** indica el valor asociado con la opción especificada; si se omite este atributo, el valor que se toma es el texto especificado para la opción. La etiqueta **option** que contenga el atributo **selected** será considerada la opción por omisión; en caso de no especificarse ninguna se considerará la primera de las opciones. Se puede también especificar el atributo **size** para indicar el número de opciones que la lista visualizará a la vez.

Para permitir realizar selecciones múltiples utilizando las teclas *Ctrl*. o *Alt*, hay que añadir el atributo **multiple**; en este caso se mostrará una lista desplegada. Por ejemplo:

<select name = “opcion” multiple >

 <option value = “1” > Opción 1

 <option selected value=”2” > Opción 2

 <option value=”3” > Opción 3

</select>

Utilizando la etiqueta optgroup, se pueden agrupar los opciones (como si de un submenú se tratara); cada grupo será identificado con una etiqueta especificada por el atributo label. Por ejemplo:

<select name = “opcion” >

 <optgroup label=”grupo 1”>

 <option value = “1” > Opción 1

 <option value = ”2” > Opción 2

 <option value = ”3” > Opción 3

 </optgroup>

 <optgroup label=”grupo 2”>

 <option value = “4” > Opción 4

 <option value = ”5” > Opción 5

 </optgroup>

</select>

Para culminar mostraremos un ejemplo en el que podemos observar varios de los comandos expuestos:

<HTML>

<HEAD>

<TITLE>Formularios - Ejemplo</TITLE>

</HEAD>

<BODY>

 <FORM name ="form2" method="POST" action="form2.php">

Escribe en este campo input: <INPUT type="text" name="inputarea" size="20"><BR>

click 1: <INPUT type="checkbox" name="micheckbox[]" value="1"><BR>

click 2: <INPUT type="checkbox" name="micheckbox[]" value="2"><BR>

click 3: <INPUT type="checkbox" name="micheckbox[]" value="3"><BR>

Selecciona uno:

verde <INPUT type="radio" name="miradio" value="verde" CHECKED>

amarillo <INPUT type="radio" name="miradio" value="amarillo">

azul <INPUT type="radio" name="miradio" value="azul"><BR>

Elige de esta lista:

<SELECT name="miselectbox">

<OPTION value="manzana">manzana</OPTION>

<OPTION value="pera">pera</OPTION>

<OPTION value="naranja">naranja</OPTION>

</SELECT><BR>

<TEXTAREA name="mitextarea" rows="4" cols="30"></TEXTAREA><BR>

<INPUT type="submit" name="enviar" value="Click para enviar!">

</FORM>

</BODY>

</HTML>

#   Bibliografía

[1] Fundamentos de bases de datos

 Abraham Silberschatz

 Editorial McGraw-Hill

[2] Aprenda. Desarrollo de Bases de Datos WEB Ya.

Jim Buyens

McGraw-Hill

[3] Desarrollo de Bases de Datos

Irene Luque Ruiz

Alfaomega

[4] Fundamentos y modelos de BASES de DATOS

Mario Piattini

 ALfaomega

[5] Php 4 a través de ejemplos

Abraham Gutiérrez, Ginés Bravo

Alfaomega, Ra-Ma

[6] MySQL

Larry Ullman

Prentice may

[7] MySQL para Windows y Linux

César Pérez

Alfaomega Ra-Ma

[8] PHP 5 para dummies

Janet Valade

St Editorial

[9] MySQL

Paul DuBois

New Riders

[10] JAVA 2. Interfaces gráficas y aplicaciones para Internet

Fco. Javier Ceballos

Alfaomega Ra-Ma