Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería Informática

Ingeniero Técnico Informática de Gestión

PROYECTO FINAL DE CARRERA

SERVIFOT STORE

Alumno: Jose Mohedano Marchante
Tutor: Enric Tordera Santamatilde
Director: Eliseo Marzal Calatayud

Valencia, Junio 2011

A mis padres José y Reme, mis hermanas Nuria y Cristina, a mis abuelos y demás familia, a mis amigos y a todo aquel que en algún momento de todos estos años me ha mostrado su apoyo y ha creído en mí.

Simplemente, GRACIAS.

Jose.

ÍNDICE

0. Acrónimos y abreviaturas

AJAX	Asynchronous JavaScript and XML.
4GL	Lenguaje de cuarta generación.
CSS	Cascading Style Sheets. Hojas de estilo en cascada.
DOM	Document Object Model. Modelo de Objetos del Documento.
ECMA	the European Computer Manufacturers' Association.
HTML	HyperText Markup Language.
HTTP	HyperText Transfer Protocol.
IBM	International Business Machines.
ISO	Organización Internacional para la Estandarización.
LDA	Lenguaje de definición de almacenamiento.
LDD	Lenguaje de definición de datos.
LDV	Lenguaje de definición de vistas.
LMD	Lenguaje de manipulación de datos.
JS	JavaScript.
OWL	Onyology Web Language.
PHP	Hypertext Proprocessor. Preprocesador de hipertexto.
RIA	Rich Internet Applications.
RSS	Really Simple Syndication.
SEQUEL	Structured English Query Languaje.
SQL	Structured Query Language. Lenguaje de consulta estructurado.
SGBD	Sistema Gestor de Bases de Datos. DataBase Management System.
SOAP	Simple Object Access Protocol.
SPARQL	Sparql Protocol and RDF Query Language.
UPV	Universidad Politécnica de Valencia.
URL	Uniform Resource Locator. Localizador Uniforme de Recursos.
W3C	World Wide Web Consortium.
XML	eXtensible Markup Language. Lenguaje de Marcas Extensible.
XHTML	eXtensible HyperText Markup Language.

1. Introducción

Actualmente las compras por Internet están a la orden del día, toda empresa que quiera abarcar lo máximo en su sector, necesita una página Web donde poder publicitar sus productos a la vez que comercializarlos.

Estos productos y todo lo relacionado respecto a ellos deben de gestionarse diariamente de forma que este cambio afecte directamente a la cantidad de información mostrada en la página de cara al cliente. Una de las formas de gestionar toda esta información es a través de un módulo de administración con el cual podemos realizar las operaciones necesarias para disponer de una página Web comercial actualizada.

El objetivo de este proyecto de fin de carrera, consiste en crear el módulo de administración para la página Web de la empresa Servifot S.L. Con este módulo queremos proporcionar una herramienta sencilla de utilizar a la vez que completa, para poder realizar las tareas de actualización sobre productos y categorías, el control de los clientes de la empresa y la gestión de los pedidos que puedan recibirse a través de la página Web.

Es importante señalar que el módulo de administración de la página Web es algo privado donde tan solo determinados usuarios (trabajadores de la empresa) pueden acceder con su autentificación para poder realizar las distintas tareas de mantenimiento y que todas estas tareas tienen su repercusión de cara al contenido y/o funcionalidad de la página Web ofrecida al cliente final.

2. Aplicaciones Web

Habitualmente, las aplicaciones muestran datos y se ejecutan en la misma máquina. Sin embargo, en los últimos años, se ha popularizado el desarrollo de aplicaciones con interfaz Web, esto es, controladas desde el navegador. Esto facilita el acceso a las mismas desde cualquier lugar con conexión a la red. Además, la mejora de las conexiones facilita su implantación.

2.1. Arquitectura de aplicaciones Web

El término de arquitectura de aplicación es usado en el diseño de aplicaciones, habitualmente del tipo cliente-servidor.

En el diseño físico se especifica exactamente donde se encontraran las piezas de la aplicación. En el diseño lógico o conceptual se especifica la estructura de la aplicación y sus componentes sin tomar en cuenta donde se localizará el software, hardware e infraestructura. Tales conceptos incluyen el orden de procesamiento, mantenimiento y seguimiento comunes en sistemas organizacionales.

Muchas veces se toma demasiado en cuenta el diseño físico de una aplicación. Por añadidura los desarrolladores generalmente asumen, indebidamente, que el diseño lógico se corresponde punto a punto con el diseño físico. Contrario a esto, un diseño adecuado debería permitir su implantación en varias plataformas y configuraciones. Como se puede ver, esta característica de portabilidad es un punto deseable para permitir que su aplicación sea flexible y escalable.

2.2. Modelo cliente / servidor

El paradigma cliente / servidor es uno de los más extendidos dentro de los servicios a través de red. Este término, en su más amplia definición, se usa para describir una aplicación en la cual dos o más procesos separados trabajan juntos para completar una tarea. El proceso cliente solicita al proceso servidor la ejecución de alguna acción en particular. Esta operación se conoce como proceso cooperativo, dado que dos procesos separados cooperan para completar la tarea en particular.

Figura 1: Modelo Cliente / Servidor

La *Figura 1* muestra el funcionamiento de este modelo. Se puede ver como el cliente realiza la petición al servidor, mientras que el servidor se dedica simplemente a responderle. De por si, el servidor tiene un papel pasivo por lo que necesita que un cliente le demande algo. Los principales servicios de Internet (WWW, FTP, SMTP, etc.) tienen clientes y servidores específicos, aunque en tiempos recientes se intenta integrar todo bajo una interfaz Web que es más amigable para el usuario.

Los procesos pueden o no estar en una sola máquina física. Tales procesos en una aplicación cliente / servidor pueden localizarse en la misma máquina o separados físicamente. El diseño lógico, y no el físico, es el que determina en que grado una aplicación es cliente / servidor.

2.3. Aplicaciones Web

La idea fundamental de los navegadores, browsers, es presentar documentos escritos en HTML que han obtenido de un servidor Web. Estos documentos HTML habitualmente presentan información de forma estática, sin más posibilidad de interacción con ellos

El modo de crear los documentos HTML ha variado a lo largo de la corta vida de las tecnologías Web pasando desde las primeras páginas escritas en HTML almacenadas en un fichero en el servidor Web hasta aquellas que se generan al vuelo como respuesta a una acción del cliente y cuyo contenido varia según las circunstancias.

Además, el modo de generar páginas dinámicas ha evolucionado, desde la utilización del Common Gateway Interface (CGI), hasta los servlets pasando por tecnologías tipo JavaServer Pages (JSP). Todas estas tecnologías se encuentran dentro de aquellas conocidas como Server Side, ya que se ejecutan en el servidor Web.

Con la extensión de Internet y de la Web en concreto, se han abierto infinidad de posibilidades en cuanto acceso a la información desde casi cualquier sitio. Esto representa un desafío a los desarrolladores de aplicaciones, ya que los avances en tecnología demandan cada vez aplicaciones más rápidas, ligeras y robustas que permitan utilizar la Web.

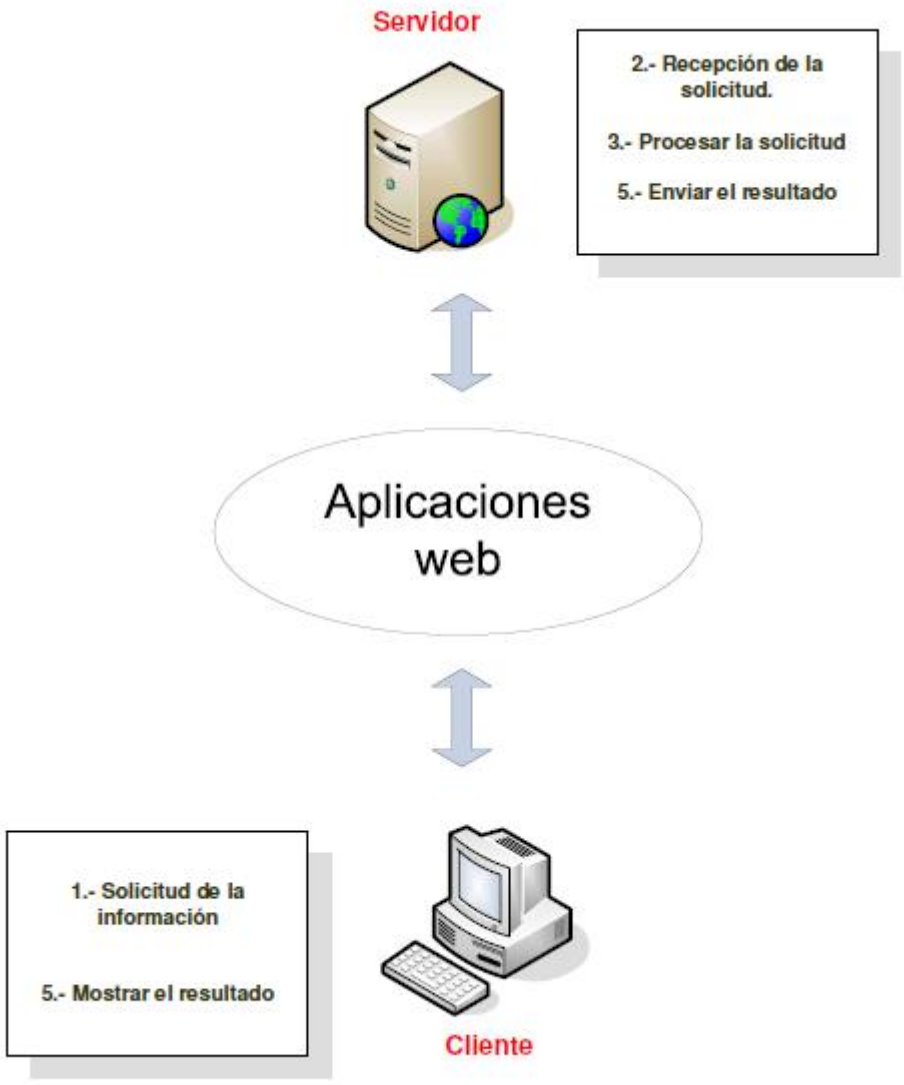


Figura 2: Pasos en el modelo cliente/servidor

Afortunadamente, se dispone de herramientas potentes para alcanzar este objetivo, ya que han surgido nuevas técnicas que permiten el acceso a una base de datos desde la Web. El único problema es decidir entre el conjunto de posibilidades la correcta para cada situación.

El protocolo CGI ha cumplido con el propósito de añadir interactividad a las páginas Web pero sus deficiencias en el desarrollo de aplicaciones y en la escalabilidad de las mismas han conducido al desarrollo de nuevas Application Programming Interface (API) específicas de servidor. Para aprovechar el potencial de estas tecnologías y ofertar una solución de servidor más extensible y portable, Sun posee la tecnología llamada servlet. Los servlets Java son eficientes, debido al esquema de threads en el que se basan y al uso de una arquitectura estándar como la Java Virtual Machine (JVM).

Otra nueva tecnología viene a sumarse a las que extienden la funcionalidad de los servidores Web, llamada JSP. Los JSP permiten unir HTML, aplicaciones Java y componentes JavaBeans creando una página Web especial que el

servidor Web compila dinámicamente en un servlet la primera vez que es llamada.

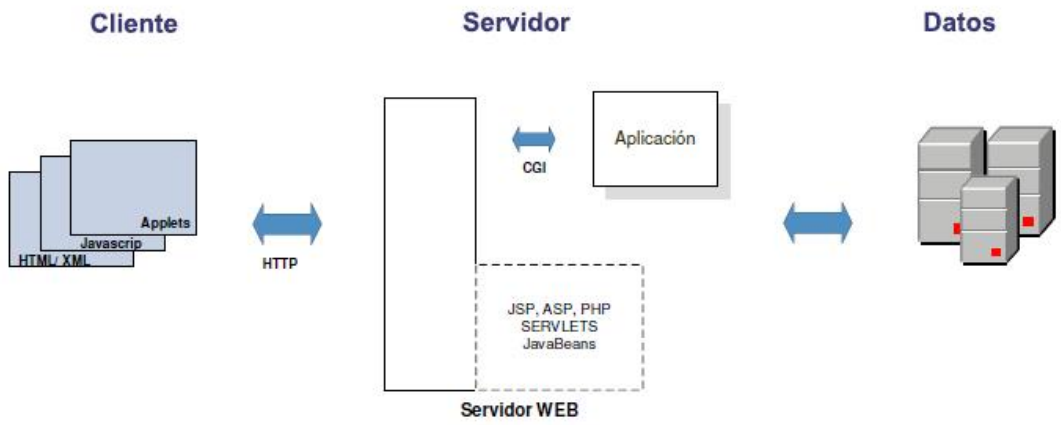


Figura 3: Esquema general de las tecnologías Web

Otro aspecto que completa el panorama son, las inclusiones del lado del cliente, Client Side, que se refieren a las posibilidades de que las páginas lleven incrustado código que se ejecuta en el cliente, como por ejemplo JavaScript.

El esquema general de la situación se puede ver en la *Figura 3*, donde se muestran cada tipo de tecnología involucrada en la generación e interacción de documentos Web.

2.4. Arquitectura de 3 niveles

La llamada arquitectura en 3 niveles es la más común en sistemas de información que, además de tener una interfaz de usuario, contemplan la persistencia de los datos.

Una descripción de los tres niveles sería la siguiente:

Nivel 1: Presentación ventanas, informes, etc.

Nivel 2: Lógica de la Aplicación. Tareas y reglas que gobiernan el proceso.

Nivel 3: Almacenamiento. Mecanismo de almacenamiento.

Figura 4: Arquitectura de 3 niveles

3. Tecnologías utilizadas

Esta sección describe las tecnologías empleadas para el desarrollo del proyecto. Para la creación de aplicaciones Web se utilizan múltiples lenguajes. En este caso se han utilizado diferentes lenguajes de programación Web como pueden ser HTML, PHP o JS.

3.1. MySQL

Figura 5: Logotipo MySQL

El sistema gestor de bases de datos es MySQL.

Se tiene por tanto, una base de datos relacional. Una base de datos relacional representa un conjunto de datos que están almacenados en tablas entre las cuales se establecen unas relaciones para manejar los datos de una forma eficiente y segura. Para usar y gestionar una base de datos relacional se usa el lenguaje estándar de programación SQL.

MySQL es Open Source, lo que significa que el código fuente se puede descargar y está accesible a cualquiera, por otra parte, usa la licencia GPL para aplicaciones no comerciales.

Además MySQL proporciona rapidez, seguridad y facilidad de manejo. Gracias a la colaboración de muchos usuarios, el sistema gestor de base de datos se ha ido mejorando optimizándose en velocidad. Por eso es uno de los sistemas gestor de bases de datos más usados en aplicaciones Web.

3.2. Hyper Text Markup Language (HTML)

Figura 6: HTML

El Hyper Text Markup Languaje (HTML), es un lenguaje de marcado, diseñado para estructurar textos y definir su presentación en forma de hipertexto, que es

el formato estándar de las páginas Web. Gracias a Internet y a los navegadores el HTML se ha convertido en uno de los formatos más populares que existen para la construcción de documentos.

Contrariamente a otros lenguajes de programación, el HTML utiliza etiquetas o marcas, que consisten en breves instrucciones de comienzo y final, mediante las cuales se determina la forma con la que deben aparecer el texto, así como las imágenes y demás elementos, en la pantalla del navegador.

Aprovechando que nos encontramos en el apartado de HTML, base indiscutible para las páginas webs, vamos a comentar varios aspectos sobre la evolución en cuanto a la forma de ver o el concepto que se tiene sobre una página Web. Aunque no son propiamente tecnologías hablaremos sobre Web 2.0 y Web 3.0.

3.2.1. Web 2.0

La Web 2.0 es la representación de la evolución de las aplicaciones tradicionales hacia aplicaciones Web enfocadas al usuario final. La Web 2.0 es una actitud y no precisamente una tecnología.

El término Web 2.0 está comúnmente asociado con aplicaciones Web que facilitan el compartir información, la interoperabilidad, el diseño centrado en el usuario y la colaboración en la World Wide Web.

Aunque el término sugiere una nueva versión de la World Wide Web, no se refiere a una actualización de las especificaciones técnicas de la Web, sino más bien a cambios acumulativos en la forma en la que desarrolladores de software y usuarios finales utilizan la Web.

La Web 2.0 es la transición que se ha dado de aplicaciones tradicionales hacia aplicaciones que funcionan a través de la Web enfocadas al usuario final. Se trata de aplicaciones que generen colaboración y de servicios que reemplacen las aplicaciones de escritorio.

Tecnologías que dan vida a un proyecto Web 2.0:

- Transformar software de escritorio hacia la plataforma del Web.
- Respeto a los estándares como el XHTML.
- Separación de contenido del diseño con uso de hojas de estilo.
- Sindicación de contenidos.
- Ajax (JavaScript asincrónico y xml).
- Uso de Flash, Flex o Lazlo.
- Uso de Ruby on Rails para programar páginas dinámicas.
- Utilización de redes sociales al manejar usuarios y comunidades.
- Dar control total a los usuarios en el manejo de su información.
- Proveer APis o XML para que las aplicaciones puedan ser manipuladas por otros.
- Facilitar el posicionamiento con URL sencillos.

Diferencias con la Web 1.0

En la Web 1.0 el usuario tenía acceso a la información solamente como receptor, no tenía la posibilidad de participar de los contenidos, las páginas eran estáticas, generalmente solo de texto y pocas imágenes, y el formato utilizado era el HTML. La interacción de los usuarios no era posible con esta forma de diseño de páginas, la información en la Web era construida solo por los dueños de los sitos, y no nutrida por las opiniones y recursos aportados por los usuarios y no se podían compartir las novedades acerca de temas de interés, es decir compartir información.

Comparación con la Web Semántica

En ocasiones se ha relacionado el término Web 2.0 con el de Web semántica. Sin embargo ambos conceptos, corresponden más bien a estados evolutivos de la Web, y la Web semántica correspondería en realidad a una evolución posterior, a la Web 3.0 o Web inteligente.

Por tanto podemos identificar la Web semántica como una forma de Web 3.0. Existe una diferencia fundamental entre ambas versiones de Web (2.0 y semántica) y es el tipo de participante y las herramientas que se utilizan. La 2.0 tiene como principal protagonista al usuario humano que escribe artículos en su blog o colabora en un Wiki. El requisito es que además de publicar en HTML emita parte de sus aportaciones en diversos formatos para compartir esta información como son los RSS, ATOM, etc. mediante la utilización de lenguajes estándares como el XML. La Web semántica, sin embargo, está orientada hacia el protagonismo de procesadores de información que entiendan de lógica descriptiva en diversos lenguajes más elaborados de metadatos como SPARQL, POWDER u OWL que permiten describir los contenidos y la información presente en la Web, concebida para que las máquinas "entiendan" a las personas y procesen de una forma eficiente la avalancha de información publicada en la Web.

Conclusión

Los proyectos tienen que renovarse y evolucionar. La Web 2.0 no es precisamente una tecnología, sino es la actitud con la que debemos trabajar para desarrollar en Internet. Tal vez allí está la reflexión más importante de la Web 2.0.

3.2.2. Web 3.0

Web 3.0 es un neologismo que se utiliza para describir la evolución del uso y la interacción en la red a través de diferentes caminos. Ello incluye, la transformación de la red en una base de datos, un movimiento hacia hacer los contenidos accesibles por múltiples aplicaciones non-browser, el empuje de las tecnologías de inteligencia artificial, la Web semántica, la Web Geoespacial, o la Web 3D. Frecuentemente es utilizado por el mercado para promocionar las mejoras respecto a la Web 2.0.

Si bien, en general, se asocia el término al de Web Semántica, acuñado por Tim Berners-Lee, cabe acotar, valga la paradoja, que no existe total consenso acerca de lo que significa la Web 3.0. Aunque se coincide en que esta etapa añadirá significado a la Web, no hay acuerdo sobre cuales son los caminos más apropiados para su desarrollo.

El aumento de la interactividad y de la movilidad son dos factores que muchos señalan como decisivos en esta nueva etapa de la Web.

¿Cómo se caracteriza y diferencia la Web 3.0 de la Web 1.0 y de la Web 2.0?

El desarrollador uruguayo, *Andrés Richero*, presenta el siguiente esquema:

- Web 1.0 – Personas conectándose a la Web.
- Web 2.0 – Personas conectándose a personas – redes sociales, Wiki, colaboración, posibilidad de compartir.
- Web 3.0 – Aplicaciones Web conectándose a aplicaciones Web, a fin de enriquecer la experiencia de las personas, a esto agrega: estado de conciencia del contexto en la Web Geoespacial, autonomía respecto del navegador y construcción de la Web Semántica.

Sobre este último punto, cabe acotar que si bien diferentes factores se conjugan en la Web 3.0 a favor de la semantización de la Web, en tanto esto es un proceso, no es privativo de ella. Por tal motivo, quizás sea más apropiado concebir la construcción de la Web Semántica, por fuera y por encima de estos estadios o instancias.

3.3. Cascading Style Sheets (CSS)

Figura 7: Logotipo CSS

Las hojas CSS son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML. El World Wide Web Consortium (W3C) es el encargado de formular la especificación de las hojas de estilo que servirá de estándar para los navegadores.

La idea que se encuentra detrás del desarrollo del CSS es separar la estructura de un documento de su presentación. Por ejemplo, el elemento de HTML H1 indica que un bloque de texto es un encabezamiento y que es más importante

que un bloque etiquetado como H2. Cuando se utiliza CSS, la etiqueta H1 no debería proporcionar información sobre como va a ser visualizado, solamente marca la estructura del documento. La información de presentación se proporciona separada en una hoja de estilo con la que se especifica como se ha de mostrar: color, fuente, alineación del texto y tamaño, además puede ser adjuntada tanto como un documento separado o en el mismo documento HTML.

Las ventajas de usar CSS son:

- Control centralizado de la presentación de un sitio Web completo, con lo que se agiliza de forma considerable la actualización del mismo.

- Los navegadores permiten a los usuarios especificar su propia hoja de estilo local, que será aplicada a un sitio Web remoto, con lo que aumenta considerablemente la accesibilidad. Por ejemplo, personas con deficiencias visuales podrían configurar su propia hoja de estilo para aumentar el tamaño del texto o remarcar más los enlaces.

- Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestre o incluso a elección del usuario. Por ejemplo, para ser impresa o mostrada en un dispositivo móvil.

- El documento HTML en sí mismo es más claro de entender y se consigue reducir considerablemente su tamaño.

Los navegadores modernos implementan CSS de forma completa, aunque existen pequeñas diferencias de implementación dependiendo del navegador.

3.4. JavaScript (JS)

Figura 8: Logotipo JavaScript

JavaScript es un lenguaje interpretado orientador a las páginas Web basado en el paradigma prototipo, con una sintaxis semejante a la del lenguaje Java.

El lenguaje JavaScript se integra dentro del código HTML de las páginas Web y actúa en cuanto un evento (un click en un botón por ejemplo) es ejecutado.

El lenguaje que fue inventado por Brendan Eich en la empresa Netscape Communications, apareció por primera vez en el Netscape Navigator 2.0.

Tradicionalmente, se venía utilizando en páginas Web HTML, para realizar tareas y operaciones en el marco de la aplicación cliente / servidor.

Los autores inicialmente lo llamaron Mocha, y más tarde, LiveScript, pero fue rebautizado como JavaScript en un anuncio conjunto entre Sun Microsystems y Netscape, en 1995.

En 1997, los autores propusieron JavaScript para que fuera adoptado como estándar internacional the European Computer Manufacturers' Association (ECMA). En junio de 1997, fue adoptado como un estándar ECMA, con el nombre de ECMAScript. Poco después también lo fue como un estándar ISO.

Para evitar estas incompatibilidades, el World Wide Web Consortium (W3C) diseñó el estándar Document Object Model (DOM), que incorporan los navegadores actuales.

3.5. Hyper Text Preprocessor (PHP)

Figura 9: Logotipo PHP

PHP es un lenguaje Open Source interpretado de alto nivel, especialmente pensado para desarrollos Web usado para generar páginas HTML. La mayoría de su sintaxis es similar a C, Java y Perl y es fácil de aprender. La meta de este lenguaje es permitir desarrollo de páginas Web, páginas dinámicas de una manera rápida y fácil, aunque su versatilidad hace que pueda emplearse en muchos otros ámbitos.

En el invierno de 1988, poco después del lanzamiento oficial de PHP 3.0, Andi Gutmans y Zeev Suraski comenzaron a trabajar en la reescritura del núcleo de PHP. Los objetivos de diseño fueron mejorar la ejecución de aplicaciones complejas y la modularidad del código.

El nuevo motor, apodado Motor Zend (comprimido de sus apellidos, Zeev y Andi), alcanzó estos objetivos de diseño satisfactoriamente, y se introdujo por primera vez a mediados de 1999.

PHP 4.0, basado en este motor, fue oficialmente liberado en mayo de 2000. Esta versión incluye otras características clave como el soporte para la mayoría de los servidores Web y sesiones HyperText Transfer Protocol (HTTP).

Hoy en día, se estima que PHP es usado por cientos de miles de programadores y muchos millones de sitios informan que lo tienen instalado, sumando más del 20% de los dominios en Internet. Además su equipo de desarrollo incluye docenas de programadores, trabajando en el núcleo o en proyectos relacionados con PHP como PEAR y el proyecto de documentación.

Actualmente tiene soporte para cosas tan dispares como Simple Object Access Protocol (SOAP), eXtensible Markup Languaje (XML), Sockets, DB, OpenSSL o llamadas POSIX.

3.6. Asynchronous JavaScript And XML (AJAX)

Figura 10: Logotipo AJAX

Ajax es una técnica de desarrollo Web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se requieren al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

3.7. Structured Query Languaje (SQL)

Figura 11: SQL

La historia de SQL, empieza en 1974 con la definición por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de International Business Machines (IBM), de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional.

Este lenguaje se llamaba Structured English Query Languaje (SEQUEL) cuyo primer prototipo, llamado SEQUEL-XRM, fue implementado entre 1974 y 1975. Los experimentos con ese prototipo condijeron, en los años siguientes a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambio de nombre por motivos legales, convirtiéndose en SQL.

El prototipo (System R), basado en este lenguaje, se adoptó y utilizó internamente en IBM y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL.

A partir de 1981, IBM comenzó a entregar sus productos relacionales y en 1983 empezó a vender DB1. En el curso de los años ochenta, numerosas compañías (por ejemplo Oracle y Sybase) comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho por lo que respecta a las bases de datos relacionales.

En 1986, el ANSI adoptó SQL como estándar para los lenguajes relacionales y en 1987 se transformo en estándar ISO. Esta versión del estándar se denominó SQL/86. En los años siguientes, éste ha sufrido diversas revisiones que han conducido, primero, a la versión SQL/89 o SQL1 y, luego, a la versión SQL/92 o SQL2. Actualmente nos encontramos en la versión de SQL/99 o SQL3 aunque en los años siguientes ha sufrido algunos cambios relacionados con la incorporación de tratamiento de XML.

El lenguaje de consulta estructurado es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo efectuar consultas con el fin de recuperar (de una forma sencilla) información de interés de una base de datos, así como también hacer cambios sobre ella. Es un lenguaje de cuarta generación (4GL).

El hecho de tener un estándar definido por un lenguaje para bases de datos relacionales abre potencialmente el camino a la ínter comunicabilidad entre todos los productos que se basan en él. Efectivamente, en general cada productor adopta e implemente en la propia base de datos solo el corazón del lenguaje SQL (el así llamado Entry level o al máximo el Intermediate level), extendiéndolo de manera individual según la propia visión que cada cual tenga del mundo de las bases de datos.

4. Bases de datos

El término base de datos fue acuñado por primera vez en 1963, en un simposio celebrado en California. De forma sencilla, una base de datos no es más que un conjunto de información relacionada que se encuentra agrupada de forma estructurada.

El archivo por sí mismo, no constituye una base de datos, sino más bien la forma en que está organizada la información es la que da origen a la base de datos. Las bases de datos manuales pueden ser difíciles de gestionar y modificar. Por ejemplo, en una guía de teléfonos no es posible encontrar el número de un individuo si no se sabe su apellido, aunque se conozca su domicilio. Del mismo modo, en un archivo de pacientes en el que la información está ordenada por el nombre, será una tarea complicada encontrar todos los pacientes que viven en una zona determinada. Todos estos problemas se pueden resolver mediante el uso de una base de datos informatizada.

Desde el punto de vista informático, una base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulan ese conjunto de datos. Desde el punto de vista más formal, se puede definir una base de datos como un conjunto de datos estructurados, fiables y homogéneos, organizados independientemente en máquina, accesibles en tiempo real, y que son compartidos por usuarios concurrentes que tienen necesidades de información diferente y no predecibles en el tiempo.

Estas colecciones de datos cumplen las siguientes propiedades:

1. Están estructuradas independientemente de las aplicaciones y del soporte de almacenamiento que las contiene.

2. Presentan la menor redundancia posible.

3. Pueden ser compartidos por varios usuarios y/o aplicaciones.

4.1. Sistema Gestor de Bases de Datos (SGBD)

Los sistemas de gestores de bases de datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos (almacenamiento físico en disco) y el usuario, o las aplicaciones que la utilizan. Se compone de:

Lenguaje de definición de datos. Una vez finalizado el diseño de una base de datos y escogido un SGBD para su implementación, el primer paso consiste en especificar el esquema lógico, el esquema interno de la base de datos, y la correspondencia entre ambos. En muchos SGBD no se mantiene una separación estricta de niveles, por lo que el administrador de la base de datos y

los diseñadores utilizan el mismo lenguaje para definir ambos esquemas: el Lenguaje de definición de datos (LDD). El SGBD posee un compilador de LDD cuya función consiste en procesar las sentencias del lenguaje para identificar las descripciones de los distintos elementos de los esquemas y almacenar la descripción del esquema en el catálogo o diccionario de datos. Se dice que el diccionario contiene metadatos: describe los objetos de la base de datos.

Cuando en un SGBD hay una clara separación entre los niveles lógico e interno, el LDD sólo sirve para especificar el esquema lógico. Para especificar el esquema interno se utiliza un lenguaje de definición de almacenamiento (LDA). Las correspondencias entre ambos esquemas se pueden especificar en cualquiera de los dos lenguajes. Para tener una verdadera arquitectura de tres niveles sería necesario disponer de un tercer lenguaje, el lenguaje de definición de vistas (LDV), que se utilizaría para especificar las vistas de los usuarios y su correspondencia con el esquema conceptual.

Lenguaje de manipulación de datos. Una vez creados los esquemas de la base de datos, los usuarios necesitan un lenguaje que les permita manipular los datos de la base de datos: realizar consultas, inserciones, eliminaciones y modificaciones, Este lenguaje es el que se denomina LMD.

Hay dos tipos de Lenguaje de manipulación de datos (LMD):

- Procedurales
- No procedurales

Con un LMD procedural el usuario (normalmente será un programador) especifica qué datos se necesitan y cómo hay que obtenerlos. Esto quiere decir que el usuario debe especificar todas las operaciones de acceso a datos llamando a los procedimientos necesarios para obtener la información requerida. Estos lenguajes acceden a un registro, lo procesan y basándose en los resultados obtenidos, acceden a otro registro, que también deben procesar. De esta manera, se va accediendo a registros y se van procesando hasta que se obtienen los datos deseados.

Un LMD no procedural se puede utilizar de manera independiente para especificar operaciones complejas sobre la base de datos de forma concisa. En muchos SGBD se pueden introducir interactivamente instrucciones del LMD desde un terminal o bien desde un lenguaje de programación de alto nivel. Los LMD no procedurales permiten especificar los datos a obtener en una consulta o los datos que se deben actualizar, mediante una sola y sencilla sentencia. El usuario o programador especifica qué datos quiere obtener sin decir cómo se debe acceder a ellos. El SGBD traduce las sentencias del LMD en uno o varios procedimientos que manipulan los conjuntos de registros necesarios. Esto libera al usuario de tener que conocer cuál es la estructura física de los datos y qué algoritmos se deben utilizar para acceder a ellos.

Lenguaje de consulta. Los sistemas de bases de datos disponen de unos lenguajes: conjunto de instrucciones que de acuerdo a una sintaxis ayudan a

realizar las distintas funciones que ha de cumplir un SGBD. El usuario final no necesita normalmente tanta potencia, por ello se le da un lenguaje de manipulación con sintaxis sencilla, a veces por medio de menús. La estructura y sintaxis de todos esos tipos de lenguajes dependen de cada SGBD, pero en los SGBD relacionales, SQL es un estándar muy extendido.

E. Codd, el creador del modelo relacional, ha establecido una lista con los ocho servicios que debe ofrecer todo SGBD.

1. Un SGBD debe proporcionar a los usuarios la capacidad de almacenar datos en la base de datos, acceder a ellos y actualizarlos. Esta es la función fundamental de un SGBD y por supuesto, el SGBD debe ocultar al usuario la estructura física interna (la organización de los ficheros y las estructuras de almacenamiento).

2. Un SGBD debe proporcionar un catálogo en el que se almacenen las descripciones de los datos y que sea accesible por los usuarios. Este catálogo es lo que se denomina diccionario de datos y contiene información que describe los datos de la base de datos (metadatos). Normalmente, un diccionario de datos almacena:

    - Nombre, tipo y tamaño de los datos.
    - Nombre de las relaciones entre los datos.
    - Restricciones de integridad sobre los datos.
    - Permisos de acceso a la base de datos.
    - Esquemas externos, conceptual e interno, y la correspondencia entre los esquemas.
    - Estadísticas de utilización, tales como la frecuencia de las transacciones y el número de accesos realizados a los objetos de la base de datos.

    Alguno de los beneficios que reporta el diccionario de datos son los siguientes:

    - La información sobre los datos se puede almacenar de un modo centralizado. Esto ayuda a mantener el control sobre los datos, como un recurso que son.
    - El significado de los datos se puede definir, lo que ayudará a los usuarios a entender el propósito de los mismos.
    - La comunicación se simplifica ya que se almacena el significado exacto. El diccionario de datos también puede identificar al usuario o usuarios que poseen datos o que los acceden.

- Las redundancias y las inconsistencias se pueden identificar más fácilmente ya que los datos están centralizados.

- Se puede tener un historial de los cambios realizados sobre la base de datos.

- El impacto que puede producir un cambio se puede determinar antes de que sea implementado, ya que el diccionario de datos mantiene información sobre cada tipo de dato, todas sus relaciones y todos sus usuarios.

- Se puede establecer políticas de la seguridad.

- Se puede garantizar la integridad.

- Se puede proporcionar información para auditorías.

3. Un SGBD debe proporcionar un mecanismo que garantice que todas las actualizaciones correspondientes a una determinada transacción se realicen, o que no se realice ninguna. Una transacción es un conjunto de acciones que cambian el contenido de la base de datos. Una transacción (en el sistema informático de una empresa inmobiliaria por ejemplo) sería dar de alta a un empleado o eliminar un inmueble. Una transacción un poco más complicada sería eliminar un empleado y reasignar sus inmuebles a otro empleado. En este caso hay que realizar varios cambios sobre la base de datos. Si la transacción falla durante su realización (debido, por ejemplo, a un fallo de hardware) la base de datos quedará en un estado inconsistente. Algunos de los cambios se habrán hecho y otros no, por lo tanto, los cambios realizados deberán ser deshechos para devolver la base de datos a un estado consistente.

4. Un SGBD debe proporcionar un mecanismo que asegure que la base de datos se actualice correctamente cuando varios usuarios la están actualizando concurrentemente. Uno de los principales objetivos de los SGBD es el permitir que varios usuarios tengan acceso concurrente a los datos que comparten. El acceso concurrente es relativamente fácil de gestionar si todos los usuarios se dedican a leer datos, ya que no pueden interferir uno con otros. Sin embargo, cuando dos o más usuarios están accediendo a la base de datos y al menos uno de ellos está actualizando datos, pueden interferir de modo que se produzcan inconsistencias en la base de datos. El SGBD se debe encargar de que estas interferencias no se produzcan en el acceso simultáneo.

5. Un SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos en caso de que ocurra algún suceso que la dañe. Como se ha comentado antes, cuando el sistema falla en medio de una transacción, la base de datos se debe devolver a un estando consistente. Este fallo puede ser a causa de un fallo en un algún dispositivo hardware o un error de software, que hagan que el SGBD aborte, o puede ser a causa de que el usuario detecte un error durante la transacción y la aborte

antes de que finalice. En todos estos casos, el SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos llevándola a un estado previo consistente.

6. Un SGBD debe proporcionar un mecanismo que garantice que sólo los usuarios autorizados pueden acceder a la base de datos. La protección debe ser contra accesos no autorizados, tanto intencionados como accidentales.

7. Un SGBD debe ser capaz de integrarse con algún software de comunicación. Muchos usuarios acceden a la base de datos desde terminales. En ocasiones estos terminales se encuentran conectados directamente a la máquina sobre la que funciona el SGBD. En otras ocasiones los terminales están en lugares remotos, por lo que la comunicación con la máquina que alberga al SGBD se debe hacer a través de una red. En cualquiera de los dos casos, el SGBD recibe peticiones en forma de mensajes y responde de modo similar. Todas estas transmisiones de mensajes las maneja el gestor de comunicaciones de datos. Aunque este gestor no forma parte del SGBD, es necesario que el SGBD se pueda integrar con él para que el sistema sea viable.

8. Un SGBD debe proporcionar los medios necesarios para garantizar que tanto los datos de la base de datos, como los cambios que se realizan sobre estos datos, sigan ciertas reglas. La integridad de la base de datos requiere la validez y consistencia de los datos almacenados. Se puede considerar como otro modo de proteger la base de datos, pero además de tener que ver con la seguridad, tiene otras implicaciones. La integridad se ocupa de la calidad de los datos. Normalmente se expresa mediante restricciones, que son una serie de reglas que la base de datos no puede violar.

Además de estos ocho servicios, es razonable esperar que los SGBD proporcionen un par de servicios más:

1. Un SGBD debe permitir que se mantenga la independencia entre los programas y la estructura de la base de datos. La independencia de datos se alcanza mediante las vistas o subesquemas. LA independencia de datos física es más fácil de alcanzar, de hecho hay varios tipos de cambios que se pueden realizar sobre la estructura física de la base de datos sin afectar a las vistas. Sin embargo, lograr una completa independencia de datos lógica es más difícil.

2. Un SGBD debe proporcionar una serie de herramientas que permitan administrar la base de datos de modo efectivo. Algunas herramientas trabajan a nivel externo, por lo que habrán sido producidas por el administrador de la base de datos. Las herramientas que trabajan a nivel interno deben ser proporcionadas por el distribuidor del SGBD. Algunas de ellas son:

- Herramientas para importar y exportar datos.
- Herramientas para monitorizar el uso y el funcionamiento de la base de datos.
- Programas de análisis estadístico para examinar las prestaciones o las estadísticas de utilización.
- Herramientas para reorganización de índices.
- Herramientas para aprovechar el espacio dejado en el almacenamiento físico por los registros borrados y que consoliden el espacio liberado para reutilizarlo cuando sea necesario.

5. Seguridad

En los últimos años, se han desarrollado multitud de aplicaciones Web que acceden a bases de datos, estas aplicaciones, al estar disponibles a través de la Web son más propensas a recibir ataques que permitan acceder a modificar la base de datos, es por ello que el tema de la seguridad es de especial interés.

5.1. Autenticación

Un sistema de autenticación es un módulo de seguridad para asegurar que el usuario que visita las páginas es quien dice ser. Por supuesto, conociendo a ese usuario, se le podrá dar acceso a más aspectos de la página que si fuese un usuario desconocido o anónimo.

Una vez que se tienen los datos de autenticación (usuario y contraseña escritos en la página inicial), se hace una comprobación de los mismos y, se redirecciona al navegador a la página de la aplicación restringida, en caso de que sean correctos, o a la página del login donde se volverá a pedir el usuario y la contraseña, en caso de que sean incorrectos.

La aplicación de acceso restringido, aparte de mostrar las funcionalidades que se quieren proteger con usuario y contraseña, debe de realizar unas comprobaciones de seguridad para saber si se ha pasado con éxito el proceso de autenticación o si se está intentando acceder de manera no permitida a esa página.

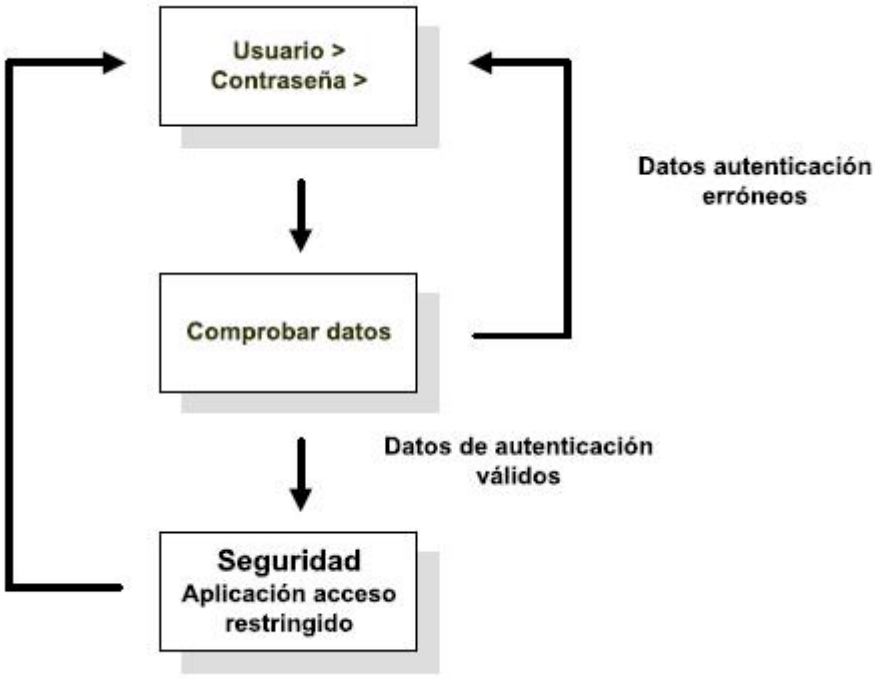


Figura 12: Autenticación de usuario

5.2. Ataques típicos

De todos los ataques, existen dos que son especialmente peligrosos: la inyección de SQL y la suplantación de identidad.

5.2.1. Suplantación de identidad

Para evitar la repetición de la contraseña, se almacenan las credenciales en la sesión del usuario en forma de token (espacio de almacenamiento temporal en el servidor). A partir de ese momento, el navegador manda un identificador de sesión (en la URL o como cookie) para usarla.

Este ataque consiste en robar la sesión de un usuario autenticado, es decir, ese identificador y usarlo para acceder a la aplicación con las credenciales del usuario.

5.2.2. Inyección de código SQL

La inyección SQL consiste en la modificación del comportamiento de las consultas mediante la introducción de parámetros no deseados en los campos a los que tiene acceso el usuario.

Este tipo de errores puede permitir a usuarios malintencionados acceder a datos a los que de otro modo no tendrían acceso y, en el peor de los casos, modificar el comportamiento de las aplicaciones.

El acceso restringido se basa en una tabla de usuarios y contraseñas y sólo los usuarios que se validen contra esa tabla podrán acceder a esos contenidos. Una manera de que los usuarios se validen será mostrar un formulario pidiendo el usuario y la contraseña y una vez que se rellenan esos campos enviar esos datos a la base de datos para comprobar si es válido.

Si el usuario escribe por ejemplo admin y de contraseña cualquier otra cosa, la sentencia no devolverá registros y no se permitirá entrar a esa persona. Pero ¿qué ocurre si el usuario escribe ‘or’1’=’1 como usuario y lo mismo de contraseña? En este caso la variable Consulta contendrá la cadena:

SELECT Count(*) FROM users WHERE login =’’ or ‘1’=’1’ AND pass = ‘’ or ‘1’=’1’

Y obviamente esta sentencia devuelve registros con lo que el usuario entrará en la Web sin tener permiso.

Pero es aún peor si el usuario utiliza estos mecanismos de inyección de SQL para ejecutar código arbitrario en el servidor: sentencias DDL, cambio de permisos, utilizar procedimientos almacenados y un largo etcétera.

Como solución, hay varios sistemas para evitarlo. Por ejemplo se puede filtrar las entradas de los usuarios reemplazando la aparición de ‘ por ‘’ (dos comillas simples) e incluso evitando que los usuarios puedan pasar caracteres como ‘ / o cualquier otro que se nos ocurra que puede causar problemas.

Otro factor importante en cuanto a la seguridad es limitar al máximo los permisos del usuario que ejecuta estas sentencias para evitar posibles problemas. Por ejemplo utilizando un usuario distinto para las sentencias SELECT, DELETE, UPDATE y asegurándonos que cada ejecución de una sentencia ejecute una sentencia del tipo permitido.

6. Estructura jerárquica

La mayor parte de usuarios alguna que otra vez han tratado con datos jerárquicos en una base de datos SQL y sin duda han aprendido que una base de datos relacional no es precisamente la mejor forma de tratar este tipo de datos.

Las tablas de una base de datos relacional no son jerárquicas (como en XML por ejemplo), son simplemente una lista plana. Los datos jerárquicos tienen una relación padre-hijo que naturalmente no está representada en una base de datos relacional.

Para nuestro objetivo, los datos jerárquicos son una colección de datos representados por nodos, donde cada uno tiene un padre y cero o más hijos (con la excepción del nodo raíz que no tiene padre). Los datos jerárquicos pueden ser encontrados en una amplia variedad de aplicaciones con bases de datos, incluyendo listas de temas en foros y correos, cuadros de organización de negocio, organización de contenido, categorías de productos, etc. Para nuestro ejemplo usaremos el siguiente esquema de categorías de producto de una ficticia tienda de electrónica:

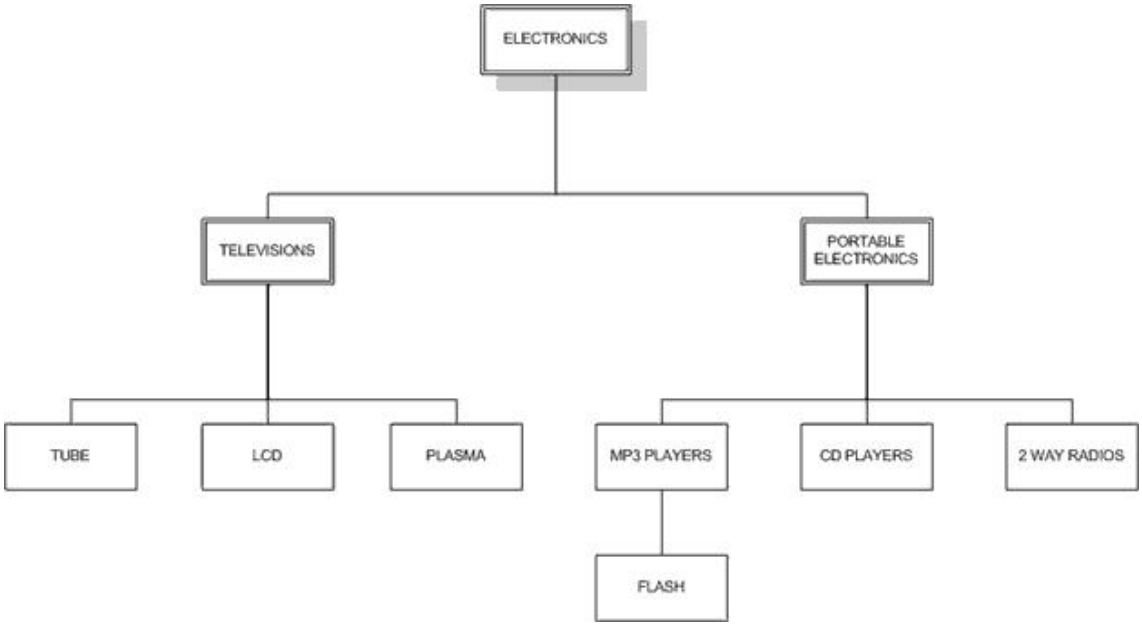


Figura 13: Esquema de categorías de producto de una tienda de electrónica ficticia

6.1. Adjacency List Model (Modelo de lista de adyacencia)

Típicamente las categorías de ejemplo mostradas anteriormente se almacenarían en una tabla de la siguiente forma:

```
CREATE TABLE category(
category_id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(20) NOT NULL,
parent INT DEFAULT NULL);


INSERT INTO category
VALUES(1,'ELECTRONICS',NULL),(2,'TELEVISIONS',1),(3,'TUBE',2),
(4,'LCD',2),(5,'PLASMA',2),(6,'PORTABLE ELECTRONICS',1),
(7,'MP3 PLAYERS',6),(8,'FLASH',7),
(9,'CD PLAYERS',6),(10,'2 WAY RADIOS',6);

SELECT * FROM category ORDER BY category_id;
```

```
+-------------+----------------------+--------+
| category_id | name                 | parent |
+-------------+----------------------+--------+
|           1 | ELECTRONICS          |   NULL |
|           2 | TELEVISIONS          |      1 |
|           3 | TUBE                 |      2 |
|           4 | LCD                  |      2 |
|           5 | PLASMA               |      2 |
|           6 | PORTABLE ELECTRONICS |      1 |
|           7 | MP3 PLAYERS          |      6 |
|           8 | FLASH                |      7 |
|           9 | CD PLAYERS           |      6 |
|          10 | 2 WAY RADIOS         |      6 |
+-------------+----------------------+--------+
10 rows in set (0.00 sec)
```

En el modelo de lista de adyacencia, cada elemento en la tabla contiene un puntero a su padre. El elemento raíz, en este caso ELECTRONICS, tiene el valor NULL en su padre. El modelo de lista de adyacencia tiene la ventaja de ser realmente muy simple, es fácil ver que la categoría FLASH es hija de MP3 PLAYERS, el cual es hijo de PORTABLE ELECTRONICS, el cual a su vez es hijo de ELECTRONICS. Mientras que el modelo de lista de adyacencia puede ser tratado fácilmente en el código del lado del cliente, trabajar con este modelo puede ser más problemático en SQL puro.

6.1.1. Recuperar un árbol entero

La primera tarea común cuando tratamos con datos jerarquizados es la visualización del árbol entero, usualmente con alguna forma de identación. La forma más común de hacer esto en SQL es usando JOIN consigo mismo:

```
SELECT t1.name AS lev1, t2.name as lev2, t3.name as lev3, t4.name as
lev4
FROM category AS t1
LEFT JOIN category AS t2 ON t2.parent = t1.category_id
LEFT JOIN category AS t3 ON t3.parent = t2.category_id
LEFT JOIN category AS t4 ON t4.parent = t3.category_id
WHERE t1.name = 'ELECTRONICS';

+-------------+----------------------+--------------+-------+
| lev1        | lev2                 | lev3         | lev4  |
+-------------+----------------------+--------------+-------+
| ELECTRONICS | TELEVISIONS          | TUBE         | NULL  |
| ELECTRONICS | TELEVISIONS          | LCD          | NULL  |
| ELECTRONICS | TELEVISIONS          | PLASMA       | NULL  |
| ELECTRONICS | PORTABLE ELECTRONICS | MP3 PLAYERS  | FLASH |
| ELECTRONICS | PORTABLE ELECTRONICS | CD PLAYERS   | NULL  |
| ELECTRONICS | PORTABLE ELECTRONICS | 2 WAY RADIOS | NULL  |
+-------------+----------------------+--------------+-------+
6 rows in set (0.00 sec)
```

6.1.2. Encontrar todos los nodos hoja

Podemos encontrar todos los nodos hoja en nuestro árbol (aquellos que no tienen hijos) usando una consulta con LEFT JOIN:

```
SELECT t1.name FROM
category AS t1 LEFT JOIN category as t2
ON t1.category_id = t2.parent
WHERE t2.category_id IS NULL;
```

```
+--------------+
| name         |
+--------------+
| TUBE         |
| LCD          |
| PLASMA       |
| FLASH        |
| CD PLAYERS   |
| 2 WAY RADIOS |
+--------------+
```

6.1.3. Recuperar un camino simple

Usar JOIN consigo mismo nos muestra también el camino completo de nuestra jerarquía:

```
SELECT t1.name AS lev1, t2.name as lev2, t3.name as lev3, t4.name as
lev4
FROM category AS t1
LEFT JOIN category AS t2 ON t2.parent = t1.category_id
LEFT JOIN category AS t3 ON t3.parent = t2.category_id
LEFT JOIN category AS t4 ON t4.parent = t3.category_id
WHERE t1.name = 'ELECTRONICS' AND t4.name = 'FLASH';
```

```
+-------------+----------------------+-------------+-------+
| lev1        | lev2                 | lev3        | lev4  |
+-------------+----------------------+-------------+-------+
| ELECTRONICS | PORTABLE ELECTRONICS | MP3 PLAYERS | FLASH |
+-------------+----------------------+-------------+-------+
1 row in set (0.01 sec)
```

La principal limitación de esta aproximación es que necesitamos un JOIN por cada nivel en la jerarquía y el funcionamiento naturalmente va empeorando con cada JOIN añadido.

6.1.4. Limitaciones del modelo de lista de adyacencia

Trabajar con el modelo de lista de adyacencia en SQL puede ser difícil. Antes de visualizar el camino completo de una categoría necesitamos conocer el nivel donde reside esta categoría. Adicionalmente, debemos tener especial cuidado cuando borramos algún nodo, ya que podemos dejar todo un subárbol sin

padre en el proceso (borrar la categoría de PORTABLE ELECTRONICS dejaría todo su subárbol sin padre). Algunas de estas limitaciones pueden ser llevadas a cabo con algo de código del lado del cliente o con procedimientos almacenados. Con un lenguaje procedural podemos empezar en el fondo del árbol e ir iterando hacia arriba para devolver el árbol entero o un camino simple. También podemos usar programación procedural para borrar nodos sin dejar huérfano el subárbol, promocionando un nodo hijo y luego reordenando el esto de hijos hasta el lugar del nuevo padre.

6.2. Nested Set Model (Modelo de conjuntos anidados)

Ahora vamos a centrarnos en una aproximación diferente para tratar datos jerarquizados, llamada el modelo de conjuntos anidados. En este modelo los datos los representamos como conjuntos. Nuestro ejemplo de la tienda de electrónica quedaría de la siguiente forma:

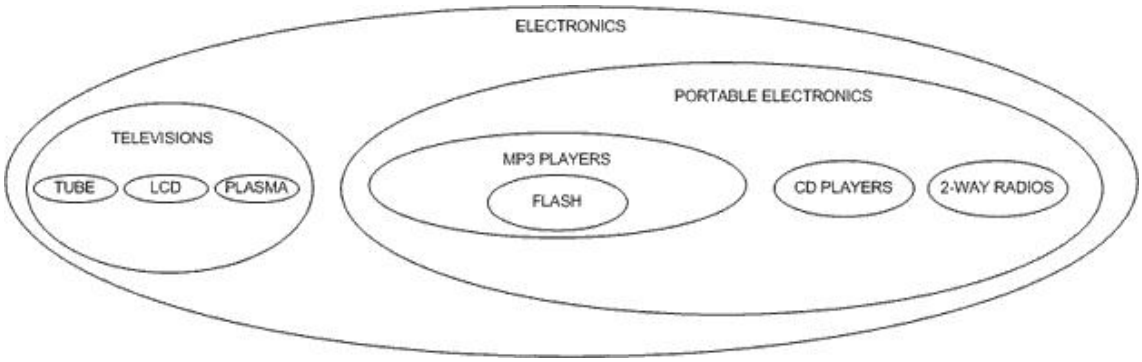


Figura 14: Esquema de conjuntos

Debemos fijarnos como la jerarquía se mantiene, como categorías de padre que envuelven a sus hijos, lo representamos en una tabla con el uso de los valores izquierda y derecha para representar la anidación de nuestros nodos:

```
CREATE TABLE nested_category (
 category_id INT AUTO_INCREMENT PRIMARY KEY,
 name VARCHAR(20) NOT NULL,
 lft INT NOT NULL,
 rgt INT NOT NULL
);


INSERT INTO nested_category
VALUES(1,'ELECTRONICS',1,20),(2,'TELEVISIONS',2,9),(3,'TUBE',3,4),
(4,'LCD',5,6),(5,'PLASMA',7,8),(6,'PORTABLE ELECTRONICS',10,19),
(7,'MP3 PLAYERS',11,14),(8,'FLASH',12,13),
(9,'CD PLAYERS',15,16),(10,'2 WAY RADIOS',17,18);


SELECT * FROM nested_category ORDER BY category_id;
```

category_id	name	lft	rgt
1	ELECTRONICS	1	20
2	TELEVISIONS	2	9
3	TUBE	3	4
4	LCD	5	6
5	PLASMA	7	8
6	PORTABLE ELECTRONICS	10	19
7	MP3 PLAYERS	11	14
8	FLASH	12	13
9	CD PLAYERS	15	16
10	2 WAY RADIOS	17	18

Usamos lft y rgt para izquierda y derecha ya que left y right son palabras reservadas.

¿Como determinamos los valores izquierda y derecha de cada elemento? Comenzamos a numerar por el lado izquierdo del nodo más exterior y continuamos por la derecha:

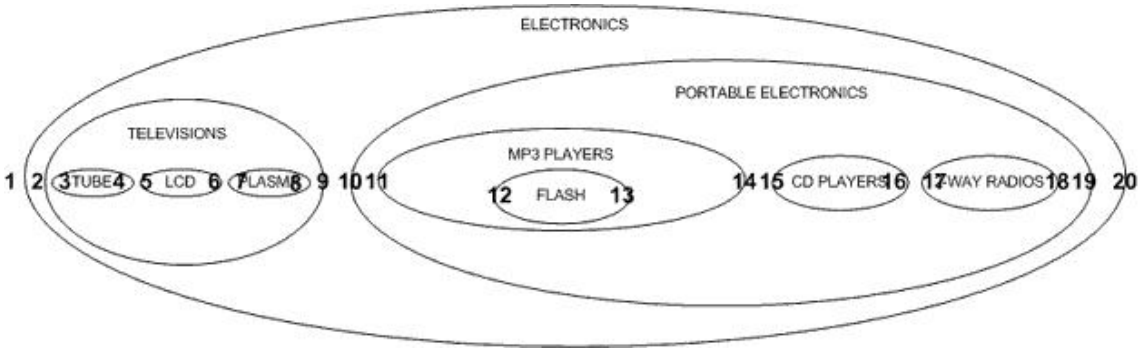


Figura 15: Esquema de conjuntos numerados

El diseño puede ser aplicado también a un árbol de nodos:

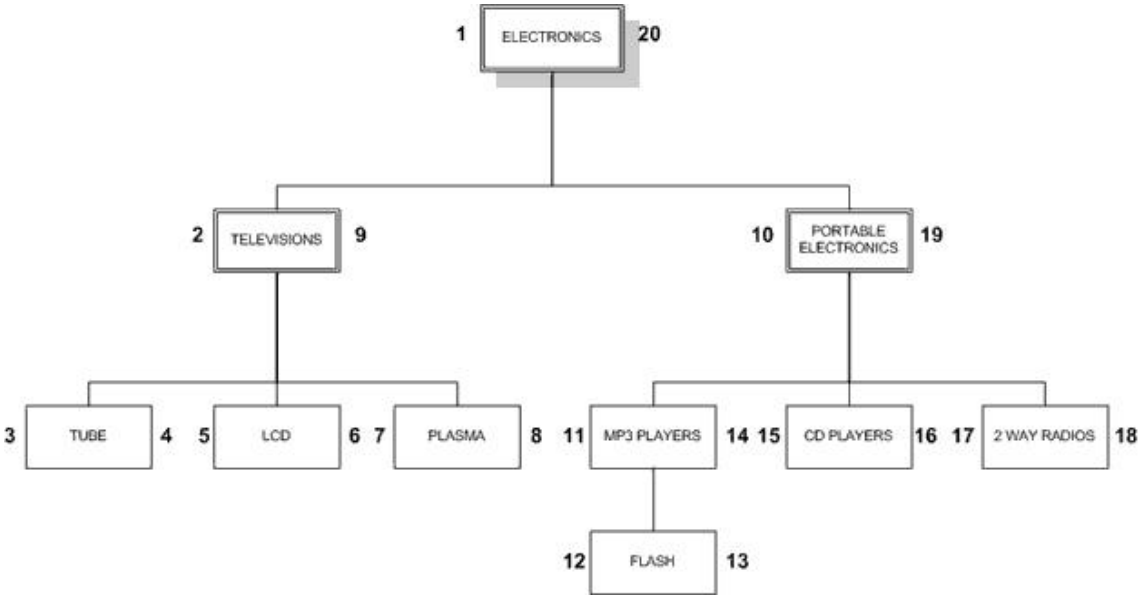


Figura 16: Árbol de nodos numerados

Cuando trabajamos con un árbol de nodos, comenzamos a numerar de izquierda a derecha, un nivel cada vez, descendiendo a cada nodo hijo antes de asignar el valor de la derecha y luego moviéndonos hacia la derecha. Esta aproximación es una modificación del algoritmo preorder tree travesal.

6.2.1. Recuperar un árbol entero

Podemos recuperar el árbol entero a través del uso de un JOIN que enlace padres con nodos partiendo de la base de que el valor izquierdo de un nodo siempre estará entre el valor izquierdo y derecho de su padre:

```
SELECT node.name
FROM nested_category AS node,
nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
AND parent.name = 'ELECTRONICS'
ORDER BY node.lft;
```

```
+----------------------+
| name                 |
+----------------------+
| ELECTRONICS          |
| TELEVISIONS          |
| TUBE                 |
| LCD                  |
| PLASMA               |
| PORTABLE ELECTRONICS |
| MP3 PLAYERS          |
| FLASH                |
| CD PLAYERS           |
| 2 WAY RADIOS         |
+----------------------+
```

Al contrario que los ejemplos anteriores vistos con el modelo de lista de adyacencia, esta consulta funcionará independientemente de la profundidad del árbol.

6.2.2. Encontrar todos los nodos hoja

Encontrar los nodos hoja en el modelo de conjuntos anidados es mucho más simple que el método LEFT JOIN usado en el modelo de lista de adyacencia. Si miramos en la tabla nested_category, podemos comprobar que los valores izquierdo y derecho para un nodo hoja son consecutivos, por lo tanto para encontrar los nodos hoja tan solo debemos mirar aquellos nodos que rgt = lft+1:

```
SELECT name
FROM nested_category
WHERE rgt = lft + 1;


+--------------+
| name         |
+--------------+
| TUBE         |
| LCD          |
| PLASMA       |
| FLASH        |
| CD PLAYERS   |
| 2 WAY RADIOS |
+--------------+
```

6.2.3. Recuperar un camino simple

Con este modelo, podemos recuperar un camino simple sin la necesidad de múltiples JOIN:

```
SELECT parent.name
FROM nested_category AS node,
nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
AND node.name = 'FLASH'
ORDER BY parent.lft;

+----------------------+
| name                 |
+----------------------+
| ELECTRONICS          |
| PORTABLE ELECTRONICS |
| MP3 PLAYERS          |
| FLASH                |
+----------------------+
```

6.2.4. Encontrar la profundidad de los nodos

Hemos visto como mostrar el árbol entero, pero si queremos ver la profundidad de cada nodo con su camino hasta llegar a él para ver como está organizado dentro del árbol no tenemos más que usar la función COUNT y la cláusula GROUP BY a la consulta que ya tenemos para poder ver la ramificación completa:

```
SELECT node.name, (COUNT(parent.name) - 1) AS depth
FROM nested_category AS node,
nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;
```

```
+----------------------+-------+
| name                 | depth |
+----------------------+-------+
| ELECTRONICS          |     0 |
| TELEVISIONS          |     1 |
| TUBE                 |     2 |
| LCD                  |     2 |
| PLASMA               |     2 |
| PORTABLE ELECTRONICS |     1 |
| MP3 PLAYERS          |     2 |
| FLASH                |     3 |
| CD PLAYERS           |     2 |
| 2 WAY RADIOS         |     2 |
+----------------------+-------+
```

Podemos usar el valor depth para indentar nuestras categorías con las funciones CONCAT Y REPEAT:

```
SELECT CONCAT( REPEAT(' ', COUNT(parent.name) - 1), node.name) AS name
FROM nested_category AS node,
nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;
```

```
+-----------------------+
| name                  |
+-----------------------+
| ELECTRONICS           |
|  TELEVISIONS          |
|   TUBE                |
|   LCD                 |
|   PLASMA              |
|  PORTABLE ELECTRONICS |
|   MP3 PLAYERS         |
|    FLASH              |
|   CD PLAYERS          |
|   2 WAY RADIOS        |
+-----------------------+
```

Aunque lo más probable es que en una aplicación cliente usemos este valor para mostrar la jerarquía. Los programadores Web pueden iterar en el árbol

añadiendo las etiquetas y cuando el valor de depth incremente y decremente.

6.2.5. Profundidad de un subárbol

Cuando necesitamos obtener la profundidad de un subárbol, no podemos limitarnos a los nodos o tablas de padres de nuestra consulta con JOIN ya que se corrompería el resultado. En lugar de eso añadimos otro JOIN con una subconsultas para determinar la profundidad que será el nuevo punto de comienzo de nuestro subárbol:

```
SELECT node.name, (COUNT(parent.name) - (sub_tree.depth + 1)) AS depth
FROM nested_category AS node,
        nested_category AS parent,
        nested_category AS sub_parent,
        (
                SELECT node.name, (COUNT(parent.name) - 1) AS depth
                FROM nested_category AS node,
                nested_category AS parent
                WHERE node.lft BETWEEN parent.lft AND parent.rgt
                AND node.name = 'PORTABLE ELECTRONICS'
                GROUP BY node.name
                ORDER BY node.lft
        )AS sub_tree
WHERE node.lft BETWEEN parent.lft AND parent.rgt
        AND node.lft BETWEEN sub_parent.lft AND sub_parent.rgt
        AND sub_parent.name = sub_tree.name
GROUP BY node.name
ORDER BY node.lft;
```

```
+----------------------+-------+
| name                 | depth |
+----------------------+-------+
| PORTABLE ELECTRONICS |     0 |
| MP3 PLAYERS          |     1 |
| FLASH                |     2 |
| CD PLAYERS           |     1 |
| 2 WAY RADIOS         |     1 |
+----------------------+-------+
```

Esta función puede ser usada con cualquier nodo, incluso con la raíz. El valor de la profundidad es siempre relativo al nodo tratado.

6.2.6. Encontrar el subordinado inmediato de un nodo

Imaginemos que queremos mostrar en una Web de productos electrónicos una lista de nuestras categorías, cuando un usuario haga click en una de las categorías que se muestren los productos de esa categoría y se listen sus categorías inmediatas pero no el subárbol entero de categorías que cuelgan de dicha categoría. Para esto, necesitamos mostrar el nodo y sus nodos inmediatos. Por ejemplo, si mostramos la categoría de PORTABLE

ELECTRONICS, querremos mostrar también MP3 PLAYERS, CD PLAYERS y 2 WAY RADIOS, perno no FLASH.

Esto puede ser fácilmente implementado añadiendo la cláusula HAVING a la consulta anterior:

```
SELECT node.name, (COUNT(parent.name) - (sub_tree.depth + 1)) AS depth
FROM nested_category AS node,
        nested_category AS parent,
        nested_category AS sub_parent,
        (
                SELECT node.name, (COUNT(parent.name) - 1) AS depth
                FROM nested_category AS node,
                nested_category AS parent
                WHERE node.lft BETWEEN parent.lft AND parent.rgt
                AND node.name = 'PORTABLE ELECTRONICS'
                GROUP BY node.name
                ORDER BY node.lft
        )AS sub_tree
WHERE node.lft BETWEEN parent.lft AND parent.rgt
        AND node.lft BETWEEN sub_parent.lft AND sub_parent.rgt
        AND sub_parent.name = sub_tree.name
GROUP BY node.name
HAVING depth <= 1
ORDER BY node.lft;

+----------------------+-------+
| name                 | depth |
+----------------------+-------+
| PORTABLE ELECTRONICS |     0 |
| MP3 PLAYERS          |     1 |
| CD PLAYERS           |     1 |
| 2 WAY RADIOS         |     1 |
+----------------------+-------+
```

Si no queremos mostrar el nodo padre tan solo deberíamos cambiar la línea HAVING depth <= 1 por HAVING depth = 1

6.2.7. Añadir nuevos nodos

Ahora veremos como actualizar nuestro árbol añadiendo nuevos nodos, para ello recordemos nuestro diagrama de conjuntos anidados de ejemplo:

Si quisiéramos añadir un nuevo nodo entre los nodos de TELEVISIONS y PORTABLE ELECTRONICS, el nuevo nodo debería tener los valores izquierdo y derecho a 10 y 11 respectivamente y todos los nodos de su derecha verían incrementado estos valores en dos unidades. Luego añadiríamos el nuevo nodo con el valor izquierda y derecha apropiado.

```
LOCK TABLE nested_category WRITE;

SELECT @myRight := rgt FROM nested_category
WHERE name = 'TELEVISIONS';

UPDATE nested_category SET rgt = rgt + 2 WHERE rgt > @myRight;
UPDATE nested_category SET lft = lft + 2 WHERE lft > @myRight;

INSERT INTO nested_category(name, lft, rgt) VALUES('GAME CONSOLES',
@myRight + 1, @myRight + 2);

UNLOCK TABLES;
```

Podemos verificar nuestro conjunto con nuestra consulta indentada:

```
SELECT CONCAT( REPEAT( ' ', (COUNT(parent.name) - 1) ), node.name) AS
name
FROM nested_category AS node,
nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;
```

```
+-----------------------+
| name                  |
+-----------------------+
| ELECTRONICS           |
|  TELEVISIONS          |
|   TUBE                |
|   LCD                 |
|   PLASMA              |
|  GAME CONSOLES        |
|  PORTABLE ELECTRONICS |
|   MP3 PLAYERS         |
|    FLASH              |
|   CD PLAYERS          |
|   2 WAY RADIOS        |
+-----------------------+
```

Si en lugar de esto queremos añadir un nodo como nodo hijo de un padre que no tiene hijos, necesitamos modificar el procedimiento ligeramente. Vamos a añadir como ejemplo un nuevo nodo FRS debajo del nodo 2 WAY RADIOS:

```
LOCK TABLE nested_category WRITE;

SELECT @myLeft := lft FROM nested_category

WHERE name = '2 WAY RADIOS';

UPDATE nested_category SET rgt = rgt + 2 WHERE rgt > @myLeft;
UPDATE nested_category SET lft = lft + 2 WHERE lft > @myLeft;

INSERT INTO nested_category(name, lft, rgt) VALUES('FRS', @myLeft + 1,
@myLeft + 2);

UNLOCK TABLES;
```

En este ejemplo ampliamos todo hacia la derecha del valor izquierdo de nuestro nuevo nodo padre y luego colocamos el nodo a la derecha del valor izquierdo. Como podemos ver, nuestro nuevo nodo está perfectamente colocado:

```
SELECT CONCAT( REPEAT( ' ', (COUNT(parent.name) - 1) ), node.name) AS
name
FROM nested_category AS node,
nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;
```

```
+-----------------------+
| name                  |
+-----------------------+
| ELECTRONICS           |
|  TELEVISIONS          |
|   TUBE                |
|   LCD                 |
|   PLASMA              |
|  GAME CONSOLES        |
|  PORTABLE ELECTRONICS |
|   MP3 PLAYERS         |
|    FLASH              |
|   CD PLAYERS          |
|   2 WAY RADIOS        |
|    FRS                |
+-----------------------+
```

6.2.8. Borrar nodos

La última tarea básica del modelo de conjuntos anidados es el borrado de nodos. Las acciones llevadas a cabo cuando borramos un nodo dependerán de la posición ocupada en la jerarquía del nodo que queramos borrar, borrar nodos hoja es mucho más sencillo que borrar nodos con hijos ya que tenemos que ocuparnos de que los hijos no queden huérfanos.

Cuando borramos un nodo hoja, el proceso es justo el opuesto al de añadir un nuevo nodo, borramos el nodo y su amplitud la restamos de cada nodo a su derecha:

```
LOCK TABLE nested_category WRITE;

SELECT @myLeft := lft, @myRight := rgt, @myWidth := rgt - lft + 1
FROM nested_category
WHERE name = 'GAME CONSOLES';

DELETE FROM nested_category WHERE lft BETWEEN @myLeft AND @myRight;

UPDATE nested_category SET rgt = rgt - @myWidth WHERE rgt > @myRight;
UPDATE nested_category SET lft = lft - @myWidth WHERE lft > @myRight;

UNLOCK TABLES;
```

Ejecutamos nuestra consulta de árbol indentado para ver que el proceso ha funcionado correctamente sin que se corrompa la jerarquía:

```
SELECT CONCAT( REPEAT( ' ', (COUNT(parent.name) - 1) ), node.name) AS
name
FROM nested_category AS node,
nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;
```

```
+----------------------+
| name                 |
+----------------------+
| ELECTRONICS          |
|  TELEVISIONS         |
|   TUBE               |
|   LCD                |
|   PLASMA             |
|  PORTABLE ELECTRONICS |
|   MP3 PLAYERS        |
|    FLASH             |
|   CD PLAYERS         |
|   2 WAY RADIOS       |
|    FRS               |
+----------------------+
```

Esta aproximación funciona de la misma forma cuando borramos un nodo y todos sus hijos:

```
SELECT @myLeft := lft, @myRight := rgt, @myWidth := rgt - lft + 1
FROM nested_category
WHERE name = 'MP3 PLAYERS';

DELETE FROM nested_category WHERE lft BETWEEN @myLeft AND @myRight;

UPDATE nested_category SET rgt = rgt - @myWidth WHERE rgt > @myRight;
UPDATE nested_category SET lft = lft - @myWidth WHERE lft > @myRight;

UNLOCK TABLES;
```

Y una vez más comprobamos que todo ha salido de forma correcta:

```
SELECT CONCAT( REPEAT( ' ', (COUNT(parent.name) - 1) ), node.name) AS
name
FROM nested_category AS node,
nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;
```

```
+----------------------+
| name                 |
+----------------------+
| ELECTRONICS          |
|  TELEVISIONS         |
|   TUBE               |
|   LCD                |
|   PLASMA             |
|  PORTABLE ELECTRONICS|
|   CD PLAYERS         |
|   2 WAY RADIOS       |
|    FRS               |
+----------------------+
```

La otra posibilidad con la que tenemos que tratar es con el borrado de un nodo pero SIN eliminar sus hijos, en algunos casos simplemente cambiaremos el nombre del padre temporalmente hasta que encontremos un sustituto, en otros casos, los nodos hijos tendrán que ser movidos hacia arriba hasta el nivel del padre:

```
LOCK TABLE nested_category WRITE;

SELECT @myLeft := lft, @myRight := rgt, @myWidth := rgt - lft + 1
FROM nested_category
WHERE name = 'PORTABLE ELECTRONICS';

DELETE FROM nested_category WHERE lft = @myLeft;

UPDATE nested_category SET rgt = rgt - 1, lft = lft - 1 WHERE lft
BETWEEN @myLeft AND @myRight;
UPDATE nested_category SET rgt = rgt - 2 WHERE rgt > @myRight;
UPDATE nested_category SET lft = lft - 2 WHERE lft > @myRight;

UNLOCK TABLES;
```

En este caso restamos dos de todos los elementos que hay a la derecha del nodo (sin hijos tendría una anchura de dos), y uno de los nodos que son hijos (para cerrar el hueco creado por la perdida del valor izquierdo del padre). Una vez más comprobamos que todo ha quedado correctamente:

```
SELECT CONCAT( REPEAT( ' ', (COUNT(parent.name) - 1) ), node.name) AS
name
FROM nested_category AS node,
nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;
```

```
+---------------+
| name          |
+---------------+
| ELECTRONICS   |
|  TELEVISIONS  |
|   TUBE        |
|   LCD         |
|   PLASMA      |
|  CD PLAYERS   |
|  2 WAY RADIOS |
|   FRS         |
+---------------+
```

7. Sesiones en PHP

Las sesiones, en aplicaciones Web realizadas con PHP y en el desarrollo de páginas Web en general, nos sirven para almacenar información que se memorizará durante toda la visita de un usuario a una página Web. Dicho de otra forma, un usuario puede ver varias páginas durante su paso por un sitio Web y con sesiones podemos almacenar variables que podemos acceder en cualquiera de esas páginas.

Digamos que las sesiones son una manera de guardar información, específica para cada usuario, durante toda su visita. Cada usuario que entra en un sitio abre una sesión, que es independiente de la sesión de otros usuarios. En la sesión de un usuario podemos almacenar todo tipo de datos, como su nombre, productos de un hipotético carrito de la compra, preferencias de visualización o trabajo, páginas por las que ha pasado, etc. Todas estas informaciones se guardan en lo que denominamos variables de sesión.

PHP dispone de un método bastante cómodo de guardar datos en variables de sesión, y de un juego de funciones para el trabajo con sesiones y variables de sesión.

Para cada usuario, PHP internamente genera un identificador de sesión único, que sirve para saber las variables que pertenecen a cada usuario. Para conservar el identificador de sesión durante toda la visita de un usuario a una página, PHP almacena la variable de sesión en una cookie, o bien la propaga a través de la URL. Esto se puede configurar desde el archivo php.ini.

8. Inclusiones de código

Muchas veces es conveniente incluir código dentro del propio código de manera repetida a lo largo de diversas páginas de forma simple, sin duplicar el código y con el propósito de no hacer la lectura de una página demasiado pesada. Para ello se utiliza el comando de php “include_once” que nos permite incluir una página .php dentro de otra página .php,

En nuestro caso es conveniente incluir los datos de acceso a la base de datos cada vez que queremos usar esta, por lo tanto en múltiples ocasiones nos interesa incluir la página “config.php”.

[config.php]

```
<?php
define("CFG_MYSQL_SERVER", "servidor");
define("CFG_MYSQL_SCHEMA", "esquema");
define("CFG_MYSQL_USER", "usuario");
define("CFG_MYSQL_PASSWORD", "contraseña");

mysql_connect(CFG_MYSQL_SERVER,CFG_MYSQL_USER,CFG_MYSQL_PA
SSWORD);
mysql_select_db(CFG_MYSQL_SCHEMA);
mysql_set_charset('utf8');
?>
```

En lugar de incluir cada vez estas líneas de código lo que hacemos es añadir en el lugar que nos interesa de nuestra página el comando “include_once” con el nombre de la página que queremos añadir.

```
…
…
…
include_once(“config.php”);
…
…
…
```

9. Acceso al módulo de administración

Como ya hemos comentado anteriormente, es importante que controlemos el acceso al módulo de administración de forma que evitemos que cualquier usuario pueda acceder a él, para ello nuestra página acceso.php se encarga de esta tarea, debemos destacar que esta página estará incluida en todas nuestras páginas a fin de evitar el acceso indebido.

También debemos tener en cuenta que necesitamos una forma de conseguir que el usuario que quiera entrar en el módulo de administración pueda tener una forma sencilla de introducir sus datos para que puedan ser validados, para ello necesitamos una página que sea la que el usuario visualiza, ya que

acceso.php simplemente actúa por detrás realizando las funciones de validación y acceso.

Para ello creamos la página acceso_formulario.php que será la que a modo de formulario nos muestre lo siguiente:

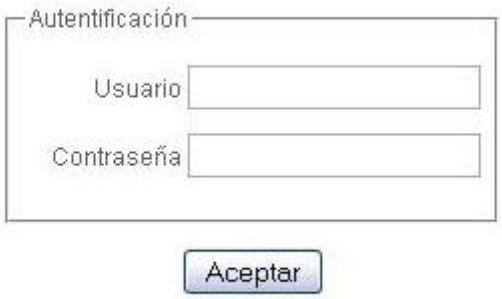


Figura 17: Formulario de acceso

El código de este formulario es el siguiente:

[acceso_formulario.php]

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
	<head>
		<meta http-equiv="Content-Type" content="text/html;
		charset=utf-8">
		<title>Autentificación</title>
		<link rel=stylesheet href="css/admin.css" type="text/css" />
	</head>
	<body>
		<form id="formulario_acceso" action="index.php" method="post"
		enctype="multipart/form-data">
			<fieldset id="datos">
				<legend>Autentificación</legend>
				<p>Usuario <input type="text" name="user"
				id="user" /></p>
				<p>Contraseña <input type="password"
				name="password" id="password" /></p>
			</fieldset>
			<p><input type="submit" value="Aceptar" /></p>
		</form>
	</body>
</html>
```

Está página está íntegramente realizada en HTML, en la etiqueta <head> tenemos el encabezado HTML que será donde configuramos el tipo de contenido mediante la etiqueta <meta>, contendrá la página y su codificación de caracteres, en nuestro caso será text/HTML y utf-8 respectivamente, en la etiqueta <title> pondremos el nombre que queramos que aparezca en la pestaña del navegador como nombre de la página, finalmente en la etiqueta

<link>, indicaremos una referencia la hoja de estilo que usaremos para darle estética a nuestra página.

Dentro de <body>, el cuerpo de la página, añadiremos nuestro formulario con la etiqueta <form>, el atributo id, asigna un identificador único para poder referenciar nuestro formulario, el atributo action se encarga de redirigir a la página que le indiquemos cuando hacemos un submit del formulario (en nuestro caso, al pulsar el botón aceptar), method es el método que usamos para enviar los datos de una página a otra. Luego creamos un <fieldset> con nombre autentificación (<legend>) que es lo que podemos visualizar en el formulario junto con dos <inputs>, de tipo text el primero donde introduciremos el usuario y de tipo password el segundo que será donde introduciremos la contraseña, estos campos también tienen el atributo id para identificarlos unívocamente. La etiqueta <p> hace referencia a un párrafo. Por último creamos el botón aceptar de tipo submit.

Por detrás de esta página, tenemos nuestra otra página de acceso, que se ejecuta en nuestro servidor y es la que se encarga de validar los datos introducidos en el formulario:

[acceso.php]

```
<?php
// Activamos el control de sesiones
session_start();

[segmento 1]
//Para cerrar la sesión
if (isset($_GET['close'])) {
        echo('<p>Sesión cerrada correctamente.</p>');
        unset($_SESSION['admin_auth']);
        unset($_SESSION['admin_nombre']);
        unset($_SESSION['admin_user']);
        unset($_SESSION['admin_type']);
        session_regenerate_id(true);
}

[segmento 2]
// Validamos usuario y contraseña
if (isset($_POST['user']) && isset($_POST['password'])) {
        if (isset($_SESSION['admin_auth'])) {
                unset($_SESSION['admin_auth']);
        }
   $login = $_POST['user'];
   $pass = $_POST['password'];

        include_once('config.php');

[segmento 3]
        $sql  = "SELECT * FROM administrators WHERE IdUser='$login'";
        $sql .= " AND Password='$pass'";
        $result = mysql_query($sql);
        if ($row = mysql_fetch_array($result)) {
                $_SESSION['admin_nombre'] = $row['Name'];
                $_SESSION['admin_auth'] = true;
                $_SESSION['admin_user'] = $login;
```

```
                $_SESSION['admin_type'] = $row['IdAccessType'];
        }
        mysql_free_result($result);
        mysql_close();
}

[segmento 4]
// Si no estamos validados mostramos el formulario de acceso
if (!isset($_SESSION['admin_auth'])) {
        readfile("acceso_formulario.php");
        die();
}
?>
```

Ahora pasaremos a explicar brevemente cada una de las partes más importantes de esta página:

En el segmento 1 del código anterior se comprueba si accedemos a la página con el atributo close y si este tiene valor, en tal caso mostramos un mensaje de sesión finalizada y eliminamos el valor que pudieran tener las variables de sesión utilizadas, además reemplazamos el id de la sesión actual.

En el segmento 2, si hemos pasado los valores user y password a través del formulario de acceso haremos que las variables $login y $pass adquieran dichos valores, además si la variable de sesión admin_auth estaba definida, le quitamos el valor que pudiera tener (unset), la destruimos.

Cuando alcanzamos el segmento 3 del código, realizamos las comprobaciones necesarias a través de una consulta SQL a la base de datos para comprobar si los datos introducidos en el formulario son válidos. Para ello primero devolveremos los datos de la tabla de administradores que se correspondan con el usuario y contraseña introducidos previamente, si existe alguno nos devolverá un registro, si no, no nos devolverá nada. Por lo tanto si nos devuelve un registro podremos crear las cuatro variables de sesión que controlaran el acceso a nuestro módulo de administración, que serán el nombre del usuario, su login, validez de su autentificación y su tipo de acceso. Posteriormente liberamos el resultado de la consulta y cerramos la conexión con nuestra base de datos.

NOTA: El tipo de acceso se usa para diferenciar entre dos niveles de usuarios administradores, los propios administradores con control total del módulo de administración y los operadores, que tienen control parcial con lo cual disponen de menos funcionalidades que un administrador.

La última parte de nuestro código (segmento 4) comprueba si la variable admin_auth está definida, en caso negativo, nos redirige al formulario de acceso y termina el script actual.

Si validamos exitosamente nuestro usuario y contraseña, entraremos a la página principal de nuestro módulo de administración y ya estaremos listos para realizar las funciones que deseemos dentro de este.

10. Servifot Store

10.1. La página principal

Figura 18: Pantalla principal del módulo de administración

Al entrar a la página principal del módulo de administración podemos observar tres partes. En la primera de ellas vemos el menú de navegación desde donde podremos acceder a las distintas secciones del módulo o desconectarnos de este,

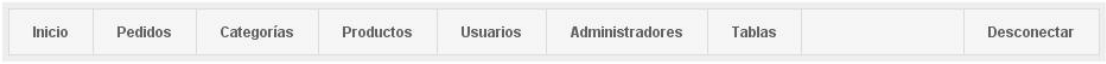


Figura 19: Menú de navegación

en la segunda parte visualizamos un resumen del estado de los pedidos actuales, separados en las tres líneas de producto que lleva la empresa y con un botón para acceder directamente a los pedidos,

Figura 20: Estado de pedidos

por último disponemos de accesos directo a determinadas tareas con el fin de agilizar algunas de las funciones que nos proporciona el módulo de administración.

Figura 21: Accesos directos

Vamos a destacar en el menú de navegación el botón desconectar que simplemente se basa en un enlace a la página acceso.php pero con la variable close a true.

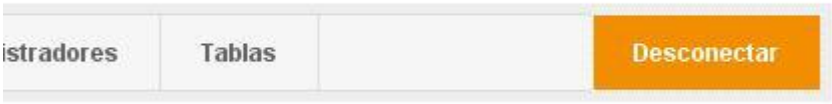


Figura 22: Botón desconectar del menú de navegación

```
http://www.servifotatl.com/admin/acceso.php?close=true
```

Si recordamos el segmento 1, visto anteriormente, incluido en la página acceso.php, podremos entender el funcionamiento, al darle valor a la variable close, la condición es cierta y se cumple, por lo tanto, todas las variables de sesión se destruyen y el identificador de la sesión se reemplaza haciendo que ya no podamos navegar por el módulo por que no estamos autentificados y volviendo otra vez al formulario de acceso.

La página de inicio, es la página principal de nuestro módulo, por lo tanto es nuestro index.php, página que se carga cuando accedemos a la carpeta admin a través de la URL del navegador.

Pasemos a ver como está estructurada:

Está página tiene un segmento de código php insertado al principio, luego en su mayoría es HTML con pequeñas inclusiones de php.

[index.php]

```
<?php
Include_once('acceso.php');
include_once('config.php');
```

```
$material_pendiente = 0;
$material_en_proceso = 0;
$pci_pendiente = 0;
$pci_en_proceso = 0;
$impresion_pendiente = 0;
$impresion_en_proceso = 0;

$sql = "SELECT IdOrderType, IdOrderState, COUNT(IdOrder) as Cantidad FROM
orders WHERE IdOrderState BETWEEN 1 AND 2 GROUP BY IdOrderType,
IdOrderState;";
$result = mysql_query($sql);

while($row = mysql_fetch_array($result)){
	if($row['IdOrderType'] == 1){
		if($row['IdOrderState'] == 1){
			$material_pendiente = $row['Cantidad'];
		}
		if($row['IdOrderState'] == 2){
			$material_en_proceso = $row['Cantidad'];
		}
	}
	if($row['IdOrderType'] == 2){
		if($row['IdOrderState'] == 1){
			$pci_pendiente = $row['Cantidad'];
		}
		if($row['IdOrderState'] == 2){
			$pci_en_proceso = $row['Cantidad'];
		}
	}
	if($row['IdOrderType'] == 3){
		if($row['IdOrderState'] == 1){
			$impresion_pendiente = $row['Cantidad'];
		}
		if($row['IdOrderState'] == 2){
			$impresion_en_proceso = $row['Cantidad'];
		}
	}
}
?>
```

En esta parte de php se incluye el código de la página de acceso (ya hemos mencionado antes que en la mayoría de las páginas está incluido para evitar intrusiones directas a nuestras distintas páginas y también se incluye el código de la página config.php, estas dos inclusiones son comunes en muchas de las páginas)

Luego se declaran seis variables que contendrán la cantidad de pedidos pendientes y en proceso de nuestras tres líneas de producto, después se realiza la consulta SQL para obtener los datos necesarios que debemos almacenar en dichas variables y por último se va iterando entre estos resultados obtenidos para asignar cada pedido devuelto a su variable adecuada.

Después de esto pasamos a ver la estructura del documento, así como php principalmente proporciona funcionalidad, HTML se ocupa de su estructura y visualización.

[index.php]

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
	<head>
		<meta http-equiv="Content-Type" content="text/html; charset=utf-
8" />
		<title>Servifot - Módulo de administración</title>
		<link rel="stylesheet" href="css/admin.css" type="text/css" />
		<link rel="stylesheet" href="css/index.css" type="text/css" />
		<script type="text/javascript" src="js/jquery-1.3.2.js"></script>
		<script type="text/javascript" src="js/index.js"></script>
	</head>
	<body>
		<div id="wrapper">
			<div id="header">
			<?php include('admin_menu.php'); ?>
			</div>
			<div id="contentHolder">
				<div id="content_special">
					<div id="main">
						<h1>Estado de pedidos</h1>
						<p>
						<div class="cuadro_resumen">
							<h1>Material
fotográfico</h1>
							<a href="#"><img
class="gotopedidos" src="images/GoToPedidos.png"/></a>
							<p><span
class="pendiente">Pendientes :</span><span class="resultado"><?php
echo($material_pendiente)?></span></p>
							<p><span
class="enproceso">En Proceso :</span><span class="resultado"><?php
echo($material_en_proceso)?></p>
						</div>
						</p>
						<p>
						<div class="cuadro_resumen">
							<h1>Productos con
imagen</h1>
							<a href="#"><img
class="gotopedidos" src="images/GoToPedidos.png"/></a>
							<p><span
class="pendiente">Pendientes :</span><span class="resultado"><?php
echo($pci_pendiente)?></span></p>
							<p><span
class="enproceso">En Proceso :</span><span class="resultado"><?php
echo($pci_en_proceso)?></p>
						</div>
						</p>
						<p>
						<div class="cuadro_resumen">
							<h1>Impresión
fotográfica</h1>
							<a href="#"><img
class="gotopedidos" src="images/GoToPedidos.png"/></a>
```

```
                                                    <p><span
class="pendiente">Pendientes :</span><span class="resultado"><?php
echo($impresion_pendiente)?></span></p>
                                                    <p><span
class="enproceso">En Proceso :</span><span class="resultado"><?php
echo($impresion_en_proceso)?></p>
                                                </div>
                                                </p>
                                                <div style="clear:
both;"> </div>
                                                <p>
                                                <h1>Accesos directos</h1>
                                                </p>
                                                <p>
                                                <div id="botones_inicio">
                                                    <a class="botonInicio"
id="iniUser" href="#"><img src="images/iniUserOff.png"/></a>
                                                    <a class="botonInicio"
id="iniAdmin" href="#"><img src="images/iniAdminOff.png"/></a>
                                                    <a class="botonInicio"
id="iniProduct" href="#"><img src="images/iniProductOff.png"/></a>
                                                    <a class="botonInicio"
id="iniShop" href="#"><img src="images/iniShopOff.png"/></a>
                                                    <a class="botonInicio"
id="iniBrand" href="#"><img src="images/iniBrandOff.png"/></a>
                                                    <a class="botonInicio"
id="iniPromo" href="#"><img src="images/iniPromoOff.png"/></a>
                                                </div>
                                                </p>
                                            </div>
                                            <div style="clear: both;"> </div>
                                        </div>
                                    </div>
                                </div>
        </body>
</html>
```

10.2. Administración de pedidos

Cuando accedemos a la administración de pedidos, lo primero que nos encontramos es esto:

Figura 23: Administración de pedidos

Cuando pulsamos sobre uno de las tres líneas de producto, la pestaña que pulsamos se marca, observamos la carga del listado, sin embargo no recargamos la página, en lugar de eso, mediante AJAX hacemos que tan solo la parte de la tabla con el listado sea la que se modifique, con lo que ahorramos tiempo y evitamos repetir recargar elementos innecesarios que incrementarían el tiempo de respuesta para el usuario.

Figura 24: Pestañas de líneas de producto y gif de carga

Una vez lista, la tabla con el listado de pedidos de esa línea de producto se nos muestra.

Figura 25: Listado de pedidos

Para facilitar la lectura del listado, agrupamos la lista en conjuntos de 10 pedidos y realizamos una paginación de la tabla, de forma que en una misma pantalla solo hay 10 pedidos y se puede leer rápidamente (esto es fácilmente modificable y podríamos agrupar los pedidos cualquier cantidad deseada). Cambiando de página vamos pasando a los demás pedidos sucesivamente.

Figura 26: Paginación

También tenemos un filtro para poder filtrar los pedidos por estado y un buscador con el que buscar pedidos por algunos de sus campos. Además de eso podemos ordenar los pedidos por determinados campos tanto ascendente como descendentemente.

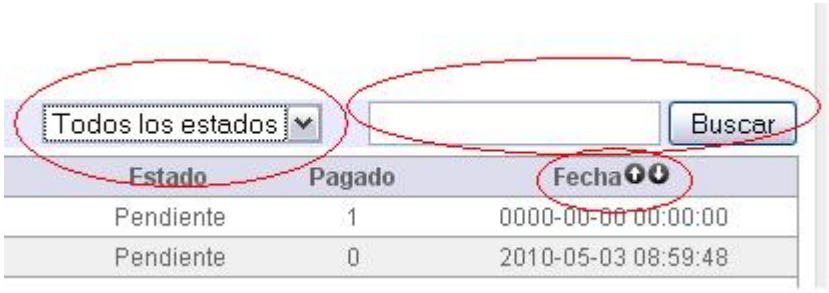


Figura 27: Filtro de estado, búsqueda y ordenación

Si pulsamos en una de las líneas de un pedido, accedemos a la página de edición de pedido, donde podemos observar y modificar determinados datos de ese pedido.

Esta sería la pantalla de edición de pedido:

Figura 28: Edición de pedido

En ella podemos observar los datos del pedido, como son su número, la fecha en la cual se efectuó, el estado del pedido, los datos de contacto del cliente (nombre, teléfono y e-mail) y tipo de pago, los datos de facturación y de envío. También podemos ver de qué productos se compone el pedido (código, descripción, cantidad, precio, total y unidades servidas) y por último un campo para las observaciones.

Respecto a la funcionalidad, podemos realizar las siguientes acciones:

- Cambiar el estado de producción del pedido
- Cambiar el estado de pago del pedido
- Servir todas las unidades de todos los productos
- No servir ninguna unidad de todos los productos
- Servir sólo algunas unidades de algún producto
- Servir todas las unidades de algún producto
- Eliminar productos del pedido
- Editar las observaciones del pedido

10.3. Administración de categorías

Figura 29: Administración de categorías

De igual forma que en la administración de pedidos, en la administración de categorías tenemos la posibilidad de decidir con que línea de producto trabajaremos a la hora de visualizar las categorías.

Al pulsar en una de las tres líneas de pedidos nos será mostrado un listado, previa carga, del árbol de categorías de esa línea de pedido.

Figura 30: Listado de categorías de material fotográfico

Debido a la cantidad de categorías y subcategorías que tenemos hemos decidido hacer que en el listado aparezcan solo las categorías principales y cuando pulsan en el nombre de una de estas, se despliega todo el subárbol de dicha categoría.

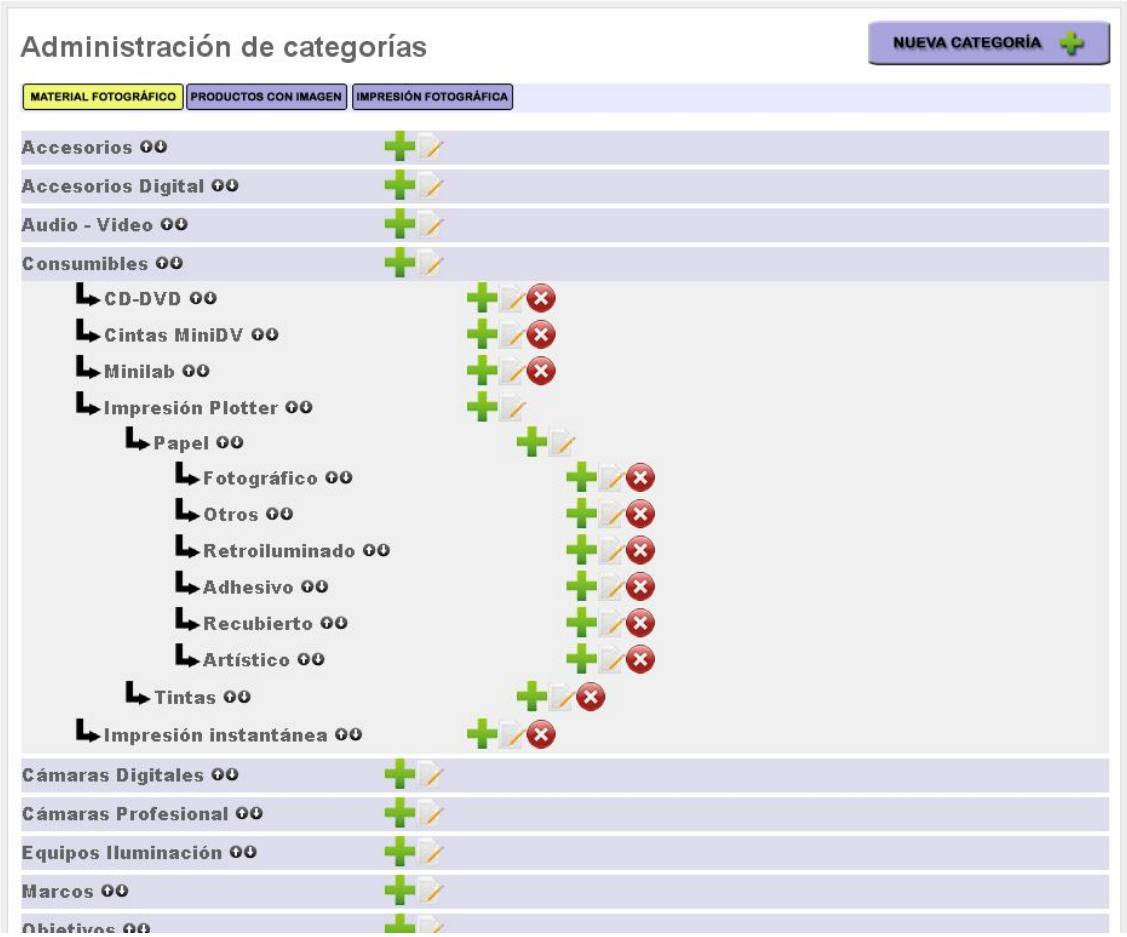


Figura 31: Subcategorías de la categoría *Consumibles*

Las funciones que podemos hacer con las categorías son:

- Mover una categoría de lugar, bien subiéndola o bajándola para poder ordenarlas como nosotros queramos.
- Añadir una categoría en el lugar que nosotros queramos (colgando de cualquier otra categoría existente).
- Editar una categoría.
- Borrar una categoría (solo si no tiene subcategorías).
- Añadir una nueva categoría.

Al añadir una nueva categoría,

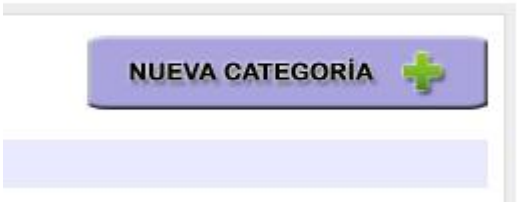


Figura 32: Botón *Nueva categoría*

accedemos al formulario de edición de categoría, donde podemos ponerle un nombre y añadir una imagen para esa categoría, que es la información que nos interesa sobre cada categoría.

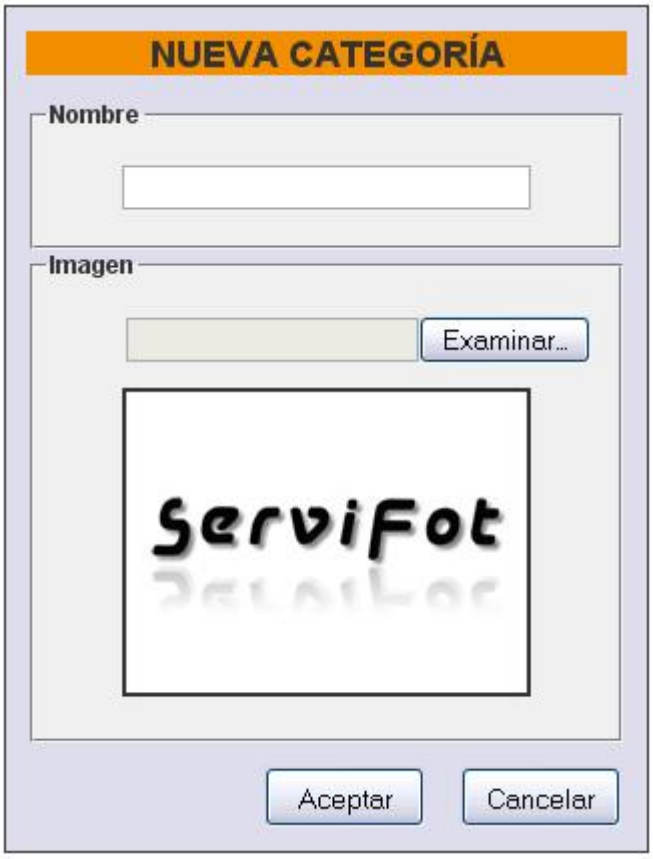


Figura 33: Edición de categoría

Toda la gestión jerárquica de las categorías se realiza mediante el modelo de conjuntos anidados visto anteriormente, debemos destacar que aunque quizá la implementación de consultas puede resultar un poco extensa, conseguimos una clara ventaja en el diseño de la base de datos, ya que tan solo con cuatro columnas, una para el identificador de la categoría, otra para su nombre y dos más para los campos izquierda y derecha tenemos todo lo necesario para representar a la perfección las categorías necesarias y poder operar con fácilmente con ellas.

10.4. Administración de usuarios

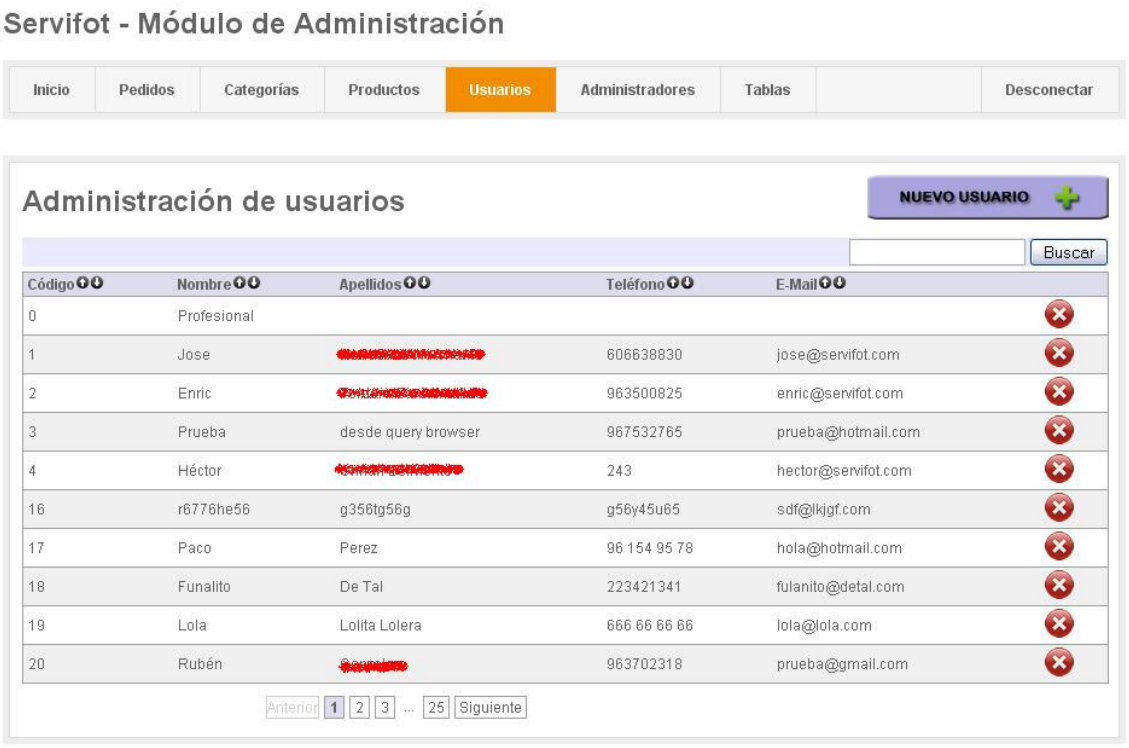


Figura 34: Administración de usuarios

La administración de usuarios es independiente de las líneas de producto, con lo que al pulsar sobre su acceso, directamente se nos carga la lista de usuarios, donde podemos ver un resumen de algunos de sus datos (los de contacto). Esta lista aparece ordenada por código de usuario pero podemos ordenarla tanto ascendente como descendentemente pulsando las flechas que aparecen al lado de cada encabezado de columna de la tabla.

Disponemos de la paginación de tabla, elemento común en la mayoría de nuestros listados.

También podemos buscar un usuario determinado usando el buscador, editar un usuario si pulsamos sobre su línea, eliminarlo si pulsamos sobre el círculo rojo con el aspa blanca que corresponde a su línea (previa confirmación de si queremos eliminarlo) o añadir un nuevo usuario, lo que nos llevaría al formulario de edición de usuario:

Figura 35: Edición de usuario

El código de usuario es un campo de solo lectura, ya que es la base de datos la que se encarga de asignarle uno automáticamente.

Entre los datos de facturación y envío podemos observar una flecha verde, si la pulsamos, todos los datos de facturación son copiados a los datos de envío, así nos ahorramos introducirlos dos veces en el caso de que sean los mismos.

Cabe destacar que hemos utilizado un plugin de jquery para realizar una validación de los datos. Más adelante hablaremos sobre este plugin.

Si hubiéramos pulsado en un usuario existente, todos los campos nos saldrían rellenos con los datos que correspondieran con dicho usuario, el formulario sería el mismo.

10.5. Administración de administradores

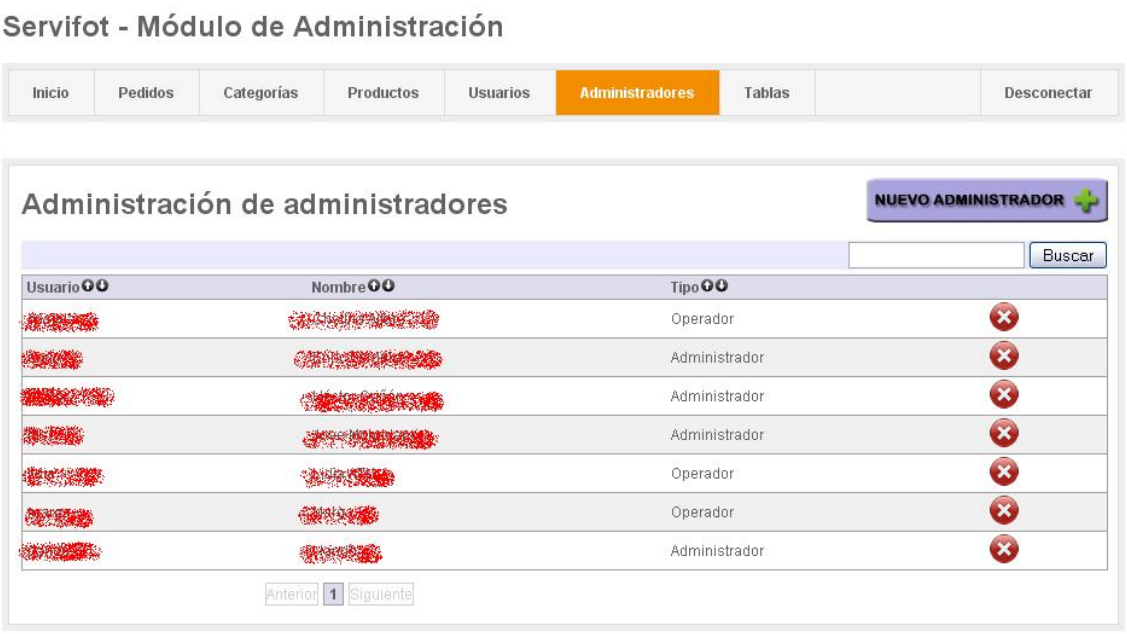


Figura 36: Administración de administradores

La vista de administración de administradores es muy parecida a la de usuarios, simplemente que en lugar de usuarios clientes de la página Web serán administradores y operadores, no hacen falta tantos datos sobre ellos como los que se necesitan de los usuarios normales.

Disponemos de las mismas funciones que en la administración de usuarios:

- Editar administradores
- Añadir administradores
- Borrar administradores
- Buscar administradores
- Paginación

El formulario de edición de administradores es mucho más simple que el de los usuarios normales:

Figura 37: Edición de administrador

El usuario y la contraseña se usarán para el acceso al módulo de administración.

El nombre se utilizará para identificar a dicho administrador / operador.

El tipo de acceso se utilizará en futuras versiones para dar distintos privilegios a operadores y administradores.

10.6. Administración de tablas

Figura 38: Administración de tablas

El apartado de tablas se realizó con la intención de poder modificar fácilmente algunas de las tablas de la base de datos a través del módulo de administración sin necesidad de tener que entrar a la base de datos cada vez que hiciera falta modificar algo de estas tablas.

Actualmente se pueden modificar tres tablas que son de bastante utilidad para la empresa, la tabla de marcas, la de tiendas y la de promociones.

10.6.1. Tabla de marcas

Figura 39: Listado de marcas

En la pestaña de marcas podemos visualizar un listado paginado con las distintas marcas con las que trabaja nuestra empresa, podemos borrar marcas, editarlas, buscarlas o añadir nuevas marcas, depende de la necesidad que tenga la empresa de trabajar con una nueva marca o de no hacerlo. Este listado condiciona las marcas que se pueden elegir al añadir o editar un nuevo producto de material fotográfico.

El formulario de edición de marcas es el más simple de todos, tan solo debemos añadir un nombre para la marca, ya que el código se introduce automáticamente.

Figura 40: Edición de marca

10.6.2. Tabla de tiendas

Figura 41: Listado de tiendas

En la pestaña de tiendas podemos visualizar el listado de tiendas con las que en algún momento ha trabajado o sigue trabajando nuestra empresa, al igual que en las demás tablas, podemos editar una tienda pulsando en su línea correspondiente, borrarla de la misma forma pulsando el icono correspondiente, buscar tiendas con el buscador, disponemos de la paginación para no hacer la lista muy grande visualmente y podemos añadir tiendas nuevas a nuestra lista, además también podemos ordenarlas según distintos campos.

Al editar o añadir una nueva tienda accedemos al formulario de edición de tienda:

Figura 42: Edición de tienda

Un campo interesante en este formulario es el de Tienda colabora, si está marcado, esta tienda aparecerá en el listado de tiendas colaboradoras de la Web principal, apareciendo todos los datos de esta (publicidad de la tienda), si no está marcado, obviamente no aparecerá en la Web principal.

10.6.3. Tabla de promociones

Figura 43: Listado de promociones

En el apartado de promociones, tenemos las promociones que la empresa va creando para usar con algunos de sus productos cada cierto tiempo, destacando el intervalo de fechas entre el cual es válido el vale y cuantos tickets se podrán crear.

Podemos realizar las mismas funciones que con las demás tablas de este apartado.

El formulario de edición de las promociones es el siguiente:

Figura 44: Edición de promoción

Existe la posibilidad de que una promoción se corresponda con más de un producto, por lo tanto en la pestaña de promoción añadimos los datos de la promoción y en la pestaña de productos:

Figura 45: Edición de productos de promoción

Podemos añadir tantos productos como entren a formar parte de dicha promoción, podemos también editar esos productos o directamente eliminarlos, también podemos añadir nuevos productos a una promoción existente.

10.7. Administración de productos

Actualmente esta sección está en desarrollo, el diseño y funcionalidad no están completamente finalizados.

Figura 46: Administración de productos

Esta es la pantalla que vemos cuando accedemos al apartado de productos, una vez más disponemos de tres pestañas para seleccionar la línea de productos con la que queremos trabajar (actualmente solo funciona la línea de material fotográfico)

Figura 47: Listado de productos

Debemos destacar el menú lateral de las categorías a través del cual podremos navegar y filtrar las distintas categorías para visualizar determinados productos.

Figura 48: Menú de categorías de productos

En este menú aparecen las categorías de productos que hemos introducido en la sección de administración de categorías para la línea de productos de material fotográfico (actualmente solo se visualiza este menú).

El formulario de edición de producto es el siguiente (actualmente en desarrollo):

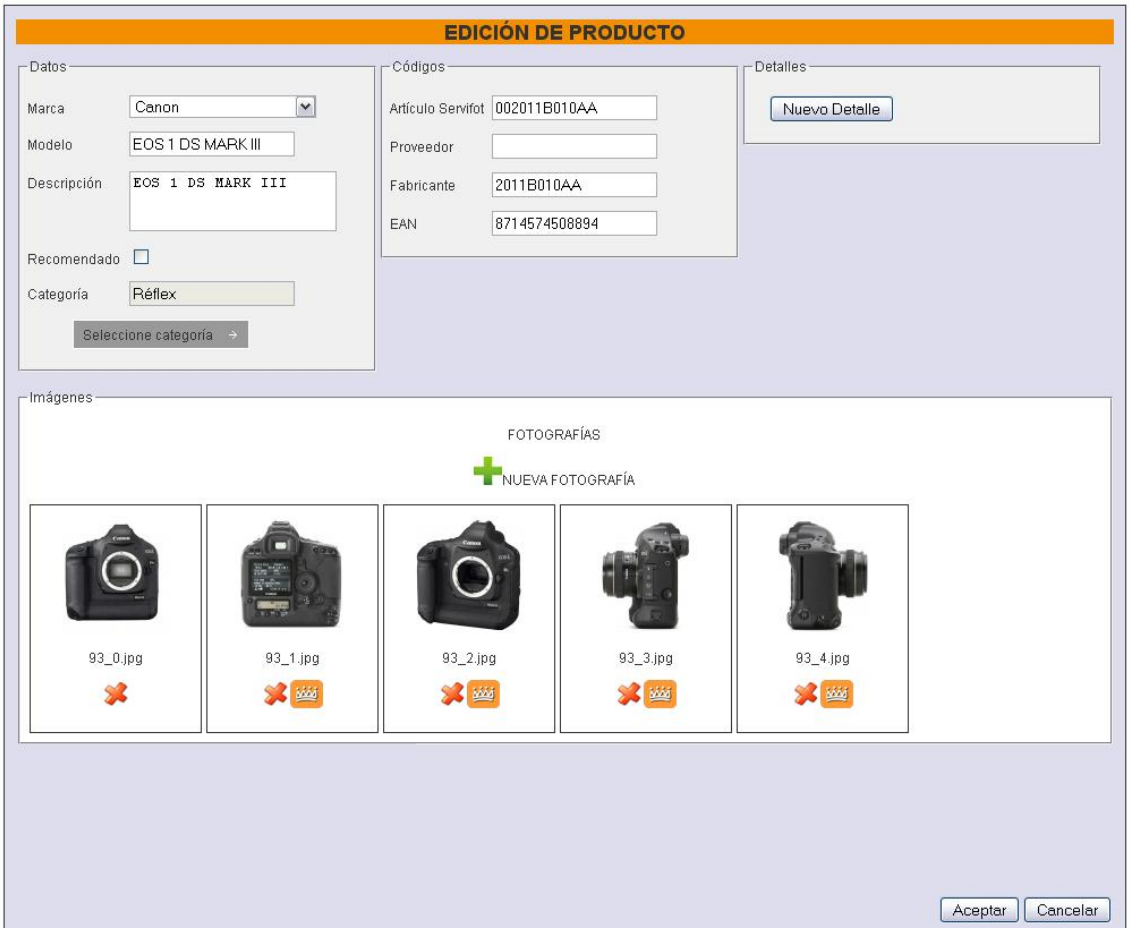


Figura 49: Edición de producto

11. Como funcionan nuestras páginas

11.1. Jquery

Para ilustrar el funcionamiento de nuestras páginas, utilizaremos como ejemplo de página de administradores, que nos sirve perfectamente ya que se puede observar el funcionamiento común de todas nuestras páginas a la hora de mostrar los datos o de realizar determinadas acciones cuando usamos determinadas funciones.

En la parte de HTML donde tenemos la etiqueta <head>, definimos los scripts que serán utilizados a lo largo de nuestra página:

```
<head>
        ...
        <script type="text/javascript" src="js/jquery-1.3.2.js"></script>
        <script type="text/javascript" src="js/administrators.js"></script>
        ...
</head>
```

Estos scripts se cargan al inicializar por primera vez la página y permanecen activos mientras esta página este cargada en nuestro navegador, es decir, cada vez que cargamos la página en cuestión.

Para todas nuestras páginas, el script jquery-1.3.2.js es común, sin embargo luego cada una tiene su propio script, en este caso administrators.js

Observemos ahora el código de administrators.js segmentado por partes y vayamos explicando cual es la función de cada una.

[administrators.js]

```
var items = 10;
var total = 0;
var page = 1;
var filter = "";
var orderby = "IdUser";

$(document).ready(function(){
        establecerBotonNuevoAdmin();
        establecerBuscar();
        establecerOrden()
        actualizarDatos();
        ponerRayasTabla();

});
```

En este segmento de código pueden diferenciarse dos partes, la primera de ellas es la definición de varias variables que serán de tipo global y que serán usadas para:

- items: cantidad de filas que tendrá la tabla donde se muestran los datos.
- total: cantidad total de datos (inicializamos a 0)
- page: página inicial (en nuestro caso es 1)
- filter: cadena por la cual se realiza una búsqueda (vacía por defecto)
- orderby: campo de la tabla por el cual ordenaremos los resultados (código de usuario [administrador] por defecto)

La segunda parte, nos muestra el evento .ready, todo lo que incluyamos en su interior será cargado justo después de cargar la página, es decir, de procesar su HTML.

Podemos observar que nosotros tenemos cinco funciones en el evento .ready que automáticamente serán ejecutadas al cargar la página de los administradores.

La forma de funcionar de jquery es a través de eventos asociados a elementos, estos elementos son identificados en la parte HTML con los atributos id y class y luego referenciados mediante jquery.

Si nos fijamos en el HTML de la página de administradores podemos comprobar como el botón (un enlace con imagen) de nuevo administrador está identificado por la clase nuevo.

[admin._administrators.php]

```
…
<a href="#"><img class="nuevo" src="images/nuevoAdminOff.png"/></a>
…
```

En el script vemos que para ese identificador existen tres eventos distintos asociados, el primero se dispara cuando pasamos el ratón por encima, el segundo cuando dejamos de pasarlo por encima y el tercero al pulsar sobre él, los eventos son .mouseover, .mouseout y .click respectivamente.

[administrators.js]

```
function establecerBotonNuevoAdmin(){
	$(".nuevo").mouseover(function(){
		var hrefOn = "images/nuevoAdminOn.png";
		$(this).attr("src", hrefOn);
	});
	$(".nuevo").mouseout(function(){
		var hrefOff = "images/nuevoAdminOff.png";
		$(this).attr("src", hrefOff);
	});
	$(".nuevo").click(function() {
		var operation = 'nuevo';
		location.href =
"admin_administrators_edit.php?operation="+operation;
	});
}
```

Al dispararse cada uno de estos eventos se realizan distintas cosas dependiendo del evento que se ejecute, para .mouseover y .mouseout lo que se hace es cambiar con el evento .attr el atributo src de ese elemento ($(this)), hacemos que cambien las imágenes entre dos y obtenemos un efecto rollover.
El evento .click lo que hace es redirigirnos a otra página mediante location.href, concretamente a la página de edición de administrador (el formulario) y además podemos añadir variables y darles valor, en nuestro caso operation=nuevo (nos servirá para saber si accedemos al formulario para añadir un nuevo administrador o para editar uno nuevo existente).

La siguiente función tiene que ver con la búsqueda de datos dentro del listado proporcionado.

[administrators.js]

```
function establecerBuscar(){
	$("#boton_buscar").click(function() {
		page = 1;
		filter = $("#filtro_busqueda").val();
		actualizarDatos();
	});
}
```

Contiene un evento .click asociado al botón buscar y hace que cuando lo pulsemos, la variable global página adquiera el valor 1, la variable filter adquiera el valor que tengamos en el cuadro de texto de buscar y luego realiza una llamada a la función actualizarDatos(), que se encargará de actualizar el listado de datos (más adelante analizaremos esta función).

La tercera función se encarga de crear los eventos para la ordenación de los datos por según que campo de la tabla queramos.

[administrators.js]

```
function establecerOrden(){
	$("#user_asc").click(function(){
		orderby = "IdUser ASC";
		filter = $("#filtro_busqueda").val();
		actualizarDatos();
	});
	$("#user_des").click(function(){
		orderby = "IdUser DESC";
		filter = $("#filtro_busqueda").val();
		actualizarDatos();
	});
	$("#name_asc").click(function(){
		orderby = "Name ASC";
		filter = $("#filtro_busqueda").val();
		actualizarDatos();
	});
	$("#name_des").click(function(){
		orderby = "Name DESC";
		filter = $("#filtro_busqueda").val();
		actualizarDatos();
	});
	$("#type_asc").click(function(){
		orderby = "Description ASC";
		filter = $("#filtro_busqueda").val();
		actualizarDatos();
	});
	$("#type_des").click(function(){
		orderby = "Description DESC";
		filter = $("#filtro_busqueda").val();
		actualizarDatos();
	});
}
```

Esta función crea eventos para las flechitas negras que aparecen a los lados de cada campo y que se usan para ordenar ascendente o descendentemente los datos por el campo que elijamos, al hacer click en ellas, se modifica la variable global orderby y filter y se llama a actualizarDatos().

Antes de pasar a comentar la función actualizarDatos(), vamos a mirar la siguiente que es una función con objetivo simplemente estético.

```
function ponerRayasTabla(){
	$('.admin_tables tr:odd').addClass("odd");
}
```

Lo que hace es asignarle a las líneas impares de la tabla la clase odd con el fin de que mediante css podamos diferenciar entre las pares y las impares y así poder darles un color diferente que mejora la lectura de los datos.

La última de nuestras funciones que se ejecutan al cargar la página de los administradores es la de actualizar los datos.

[administrators.js]

```
function actualizarDatos(){
	var url = "admin_get_administrators.php";
	$.ajax({
		type: "GET",
		url: url,
		data: "filter="+filter+"&orderby="+orderby+"&start="+((page-
1)*items)+"&numrows="+items,
		dataType: "xml",
		success: function(xml){
			clearData();
			$(xml).find('Admin').each(function(){
				var IdUser = $(this).find('IdUser').text();
				var Name = $(this).find('Name').text();
				var Description = $(this).find('Description').text();

				$("#tablaadministradores").append('<tr></tr>');
				$("#tablaadministradores tr:last").append('<td
class="td_'+IdUser+'">'+IdUser+'</td>');
				$("#tablaadministradores tr:last").append('<td
class="td_'+IdUser+'">'+Name+'</td>');
				$("#tablaadministradores tr:last").append('<td
class="td_'+IdUser+'">'+Description+'</td>');
				$("#tablaadministradores tr:last").append('<td><a
id="borrar_'+IdUser+'" href="#"><img src="images/delete.png"/></a></td>');
				establecerGoToAdmin(IdUser);
				establecerBorrarAdmin(IdUser);
				});
			ponerRayasTabla();
			$(xml).find('Count').each(function(){
				total = $(this).find('Total').text();
			});
			updatePagination();
		}
	});
}
```

Esta función utiliza AJAX para obtener los datos y posteriormente actualizarlos en la página Web sin tener que recargarla.

Creamos una variable URL, donde asignamos la URL de la página a la que accederá para obtener los datos, en nuestro caso admin_get_administrators.php (posteriormente analizaremos esta página), luego creamos la función de Ajax y le pasamos varias variables con sus valores, el type será el modo en el que se pasan los datos, GET quiere decir que los cogerá de propia URL, URL como ya hemos comentado es la página Web a la que accederá, data son los atributos adicionales que serán pasados en la URL, en nuestro caso pasaremos los valores de las variables globales que creamos al principio del todo, en dataType indicamos el tipo de los datos devueltos, que será un XML y luego en success asignamos que acciones se deben de llevar a cabo si al ejecutar el AJAX tenemos éxito y nos devuelve algún tipo de información válida, en nuestro caso como ya hemos comentado un XML.

La página admin_get_administrators.php nos devuelve un fichero con formato XML donde están contenidos los datos de los administradores que son los que necesitamos para tratarlos y mostrarlos, en esencia esto es lo que nos devuelve:

```
<AdminList>
	<Admin>
		<IdUser>Usuario 1</IdUser>
		<Name>Nombre 1</Name>
		<Description>Operador</Description>
	</Admin>
	<Admin>
		<IdUser>Usuario 2</IdUser>
		<Name>Nombre 2</Name>
		<Description>Administrador</Description>
	</Admin>
	<Admin>
		<IdUser>Usuario 3</IdUser>
		<Name>Nombre 3</Name>
		<Description>Administrador</Description>
	</Admin>
...
...
...
	<Count>
		<Total>7</Total>
	</Count>
</AdminList>
```

NOTA: Hemos cambiado los usuarios y nombres por motivos de seguridad.

Al tener éxito en nuestro AJAX, primero ejecutamos la función clearData(), esta función se encarga de borrar el texto del cuerpo de la tabla y la paginación.

[administrators.js]

```
function clearData() {
	$("#tablaadministradores").text('');
	$("#pagination").text('');
}
```

[admin_administrators.php]

```
...
<table class="admin_tables">
        <thead>
...
...
...
        </thead>
        <tbody id="tablaadministradores">
        </tbody>
</table>
<div id="pagination">
</div>
...
```

Posteriormente para cada grupo de etiquetas <admin> encontrado en el XML, realiza las siguientes acciones:

Crea tres variables y en ellas pone el valor de los tres campos que necesitamos mostrar, IdUser, Name y Description y que obtiene del XML. Luego en tiempo real añade al cuerpo de la tabla estos datos junto con el símbolo de borrar administrador y les asigna identificadores. En ese momento se llama a dos funciones más que serán las que creen los eventos para borrar un administrador (cuando hagamos click en el botón correspondiente) y para editar un administrador (cuando hagamos click en cualquier parte de su línea en la tabla).

Una vez finalizado todo el bucle, ya tendremos los datos de los administradores, entonces usamos la función ya vista para ponerle rayas a la tabla (ponerRayasTabla());

En el XML además de los datos de los administradores, al final tenemos un campo con el total de registros, campo que leeremos y actualizaremos la variable global (total) que nos será de mucha utilidad para establecer la paginación, que será la siguiente función a la que llamemos.

Primero veremos las funciones establecerGoToAdmin() y establecerBorrarAdmin() y posteriormente la función updatePagination().

La función establecerGoToAdmin() crea el evento click cuando pulsamos en cualquier parte de la fila de un administrador y hace que seamos redirigidos al formulario de edición de administrador.

[administrators.js]

```
function establecerGoToAdmin(IdUser){
	$(".td_"+IdUser).click(function(){
		var operation = 'editar';
		var IdUser = $(this).attr("class");
		IdUser = IdUser.replace("td_", "");
		location.href =
"admin_administrators_edit.php?IdUser="+IdUser+"&operation="+operation;
	});
}
```

Debemos destacar la parte en negrita y explicar por que debemos hacer eso para obtener IdUser. Cuando le pasamos a la función IdUser nos sirve para crear el evento asociado, pero dentro del evento click, no tenemos el valor de IdUser, la única forma de obtenerlo es cogerlo del identificador de clase que hemos creado anteriormente y reemplazar la parte “td_” por [cadena vacía] para así conseguir el IdUser de ese administrador en el que estamos haciendo click.

La función establecerBorrarAdmin() se encarga de crear el evento para borrar un administrador de nuestra base de datos y también de la lista actual que estamos visualizando.

[administrators.js]

```
function establecerBorrarAdmin(IdUser){
	$("#borrar_"+IdUser).click(function(){
		var IdUser = $(this).attr("id");
		IdUser = IdUser.replace("borrar_", "");
		if(quiereBorrar(IdUser)){
			borrarAdmin(IdUser);
		}
	});
}
```

Al igual que en la anterior función debemos usar el mismo método para recuperar el IdUser de un administrador, luego llamamos a la función quiereBorrar(IdUser).

[administrators.js]

```
function quiereBorrar(IdUser){
	var ok = confirm("¿Quiere borrar el usuario con ID: "+IdUser+"?");
	if(ok) return true;
	else return false;
}
```

Esta función nos saca por pantalla un cuadro preguntándonos si queremos borrar el administrador, en caso afirmativo devuelve true, en caso negativo false.

Si devolvemos true, se ejecutará la función de borrarAdmin().

[administrators.js]

```
function borrarAdmin(IdUser){
	var operation = 'borrar';
	var url = "admin_operation_administrators.php";
	$.ajax({
		type: "GET",
		url: url,
		data: "IdUser="+IdUser+"&operation="+operation,
		dataType: "xml",
		success: function(xml){
			var status = $(this).find('Status').text();
			if(status=="0"){
				var IdUser = $(this).find('UserId').text();
				alert("El usuario con ID: "+IdUser+"no ha podido
ser eliminado");
			}else{
				clearData();
				actualizarDatos();
			}
		}
	});
}
```

Con esta función volvemos a usar AJAX para realizar la tarea, esta vez llamamos a otra página donde se encuentran las operaciones que podemos tratar respecto a los administradores, admin_operation_administrators.php, esta página también nos devuelve un XML, pero en lugar de contener datos, simplemente nos devolverá true o false dependiendo de si se ha podido realizar la acción que queremos llevar a cabo o no, si el XML tiene éxito y el campo Status es false o 0, el sistema nos devolverá un mensaje diciéndonos que no se ha podido borrar el administrador, en caso contrario, se actualizará la tabla de administradores y podremos observar que el administrador borrado ya no está, todo esto en tiempo “real” mediante AJAX y sin tener que cargar contenido adicional con todo lo que ello implica.

Ahora pasemos a ver la última de nuestra funciones, la que se encarga de actualizar la paginación dependiendo de si tenemos más o menos datos que mostrar.

[administrators.js]

```
function updatePagination() {
	var numpaginas = Math.ceil(total/items);
	$("div#pagination").text('');
	$("div#pagination").append("<ul></ul>");
	if (page > 1) {
		$("div#pagination ul").append('<li><a id="-1"
href="#">Anterior</a></li>');
	}else $("div#pagination ul").append('<li class="disabled">Anterior</a>');
	if (page > 3) {
		$("div#pagination ul").append('<li><a id="1"
href="#">1</a></li>');
	}
	if (page > 4) {
		$("div#pagination ul").append('<li class="spacer">...</li>');
	}
	for (i=page-2; i<=(page+2); i++) {
		if ((i>=1) && (i<=numpaginas)) {
			if (i==page) {
				clase = ' class="current"';
			} else {
				clase = '';
			}
			$("div#pagination ul").append('<li'+clase+'><a id="'+i+'"
href="#">'+i+'</a></li>');
		}
	}
	if ((numpaginas-page) > 3) {
		$("div#pagination ul").append('<li class="spacer">...</li>');
	}
	if ((numpaginas-page) > 2) {
		$("div#pagination ul").append('<li><a id="'+numpaginas+'"
href="#">'+numpaginas+'</a></li>');
	}
	if (page < numpaginas) {
		$("div#pagination ul").append('<li><a id="+1"
href="#">Siguiente</a></li>');
	}else $("div#pagination ul").append('<li class="disabled">Siguiente</li>');

	$("div#pagination li a").click(function() {
		var newpage = $(this).attr("id");
		if (newpage == "+1") {
			page = page+1;
		} else if (newpage == "-1") {
			page = page-1;
		} else {
			page = Number(newpage);
		}
		clearData();
		actualizarDatos();
		return false;
	});
}
```

Lo primero y más importante que realiza esta función es calcular el número de páginas que deberá mostrar, esto lo hace dividiendo el total de registros obtenidos en el XML de los datos de administrador (si recordamos al final se

devolvía el total y se actualizaba la variable global total) entre el número de registros que queremos mostrar (variable global ítems).

Después de este cálculo, realiza una serie de comprobaciones para ir creando la estructura HTML que tendrá la paginación, añadiendo elementos a una lista según corresponda.

Por último, creamos el evento asociado a hacer click en un número de página o en las palabras anterior y siguiente, que hará que nuestra variable global page sea modificada y posteriormente llamamos a clearData() y actualizarDatos(), al tener la variable page otro valor y no el 1 que tiene por defecto, en la llamada de actualizar datos, cuando le pase este valor, la página que devuelve los datos de los administradores en XML, nos devolverá los datos adecuados para el número de página que estamos visualizando.

11.2. Ajax

Veamos ahora la página admin_get_administrators.php que es la que se encarga de devolver los datos de los administradores en formato XML.

Vayamos parte por parte explicando y observando que funciones realiza cada segmento de código.

Las dos primeras líneas han sido ya vistas en varias ocasiones, se encargan de la seguridad y de la configuración y conexión a la base de datos.

[admin._get_administrators.php]

```
// Seguridad
include('acceso.php');
// Establecemos la conexión con la BD
include_once('config.php');
```

Luego tenemos la función principal que crea toda la estructura XML con los campos y valores que nosotros le pasamos. Básicamente lo que hace es recorrer el resultado de la consulta a la base de datos ($result) e ir leyendo y agrupando los datos creando el XML y luego devolverlo.

[admin._get_administrators.php]

```
// Función para generar XML
function ResultToXML(&$result, $containerName="container",
$elementName="element", $encoding="Shift_JIS", $total=0) {
   //this functions creates XML output from the SQL result.
   $xml = "<?xml version=\"1.0\" encoding=\"$encoding\" ?>\n";
        $xml .= "<$containerName>\n";
   while ($stuff = mysql_fetch_assoc($result)) {
      $xml .= "  <$elementName>\n";
      foreach($stuff as $key=>$value) {
         $value = htmlspecialchars($value);
         $xml .= "    <$key>$value</$key>\n";
      }
```

```
        $xml .= "  </$elementName>\n";
    }
        if ($total != 0) {
                $xml .= "  <Count>\n";
                $xml .= "    <Total>".$total."</Total>\n";
                $xml .= "  </Count>\n";
        }
        $xml .= "</$containerName>\n";
    return $xml;
}
```

Cuando llamamos a esta página, podemos pasarle varios atributos, si no cogerá los que tenga por defecto. Aquí es donde cogerá el campo búsqueda, la ordenación y los datos para la paginación como son el número de filas que queremos mostrar en la tabla y desde que dato queremos comenzar. Esta parte se encarga de eso:

[admin._get_administrators.php]

```
// Recuperamos parámetros
if (isset($_GET["filter"])) {
        $filter = $_GET["filter"];
} else {
        $filter = "";
}
if (isset($_GET["orderby"])) {
        $orderby = $_GET["orderby"];
} else {
        $orderby = "IdUser";
}
// Recuperamos datos para la paginación
$start = 0;
$numrows = 10;
if (isset($_GET["start"])) $start = $_GET["start"];
if (isset($_GET["numrows"])) $numrows = $_GET["numrows"];
```

Posteriormente con todo estos datos realizará dos consultas SQL, en una de ellas obtendrá los datos de los administradores que necesitamos y la otra el número total de administradores. Debemos destacar la línea en negrita, por que es la que tiene que ver directamente con la paginación y la que hará que según el valor de las variables $start y $numrows, nos sean devueltos unos datos u otros.

[admin._get_administrators.php]

```
// Generamos la consulta SQL
$sql = "SELECT a.IdUser, a.Name, at.Description";
$sql .= " FROM administrators a, administrator_types at";
$sql .= " WHERE a.IdAccessType = at.IdAccessType";
if($filter != ""){
        $sql .= " AND (IdUser LIKE '%$filter%' OR Name LIKE '%$filter%')";
}
$sql .= " ORDER BY $orderby";
$sql .= " LIMIT $start,$numrows;";
// Ejecutamos la consulta
$result = mysql_query($sql);
```

```
if (!$result) die();

// Contamos el total de registros
$sql = "SELECT Count(IdUser) AS Total";
$sql .= " FROM administrators a, administrator_types at";
$sql .= " WHERE a.IdAccessType = at.IdAccessType";
if($filter != ""){
        $sql .= " AND (IdUser LIKE '%$filter%' OR Name LIKE '%$filter%')";
}
$result_count = mysql_query($sql);
$row = mysql_fetch_array($result_count);
$total = $row["Total"];
```

Una vez tenemos las dos consultas, tenemos dos variables importantes, $result y $total, en $result almacenamos el resultado de la consulta y el $total, el número total de filas de la tabla que estamos consultando (en este caso, cantidad de administradores), y entonces realizamos la llamada a la función:

[admin._get_administrators.php]

```
// Generamos XML
$xmldata = ResultToXML($result,"AdminList","Admin","UTF-8",$total);
```

En esta función pasamos, el resultado de la consulta $result, pasamos dos cadenas, que serán los nombres del XML que tendrán el contenedor global de datos y luego para cada grupo de datos otro nombre, en nuestro caso serán AdminList y Admin, una codificación de caracteres (nosotros siempre trabajamos con UTF-8) y la variable $total, donde pasamos la cantidad total.

Por último devolvemos el XML ya creado en forma de página:

[admin._get_administrators.php]

```
// Devolvemos el resultado
header('Content-type: text/xml');
header('Pragma: public');
header('Cache-control: private');
header('Expires: -1');
echo($xmldata);
```

Otra página a la que llamamos para realizar una de las operaciones es a la página admin_operation_administrators.php, esta página es muy parecida a la que acabamos de ver en cuanto a funcionalidad, le pasamos algunos valores y nos da una respuesta en forma de XML que utilizaremos para varias cosas, entre ellas comprobar si hemos borrado un administrador.

[admin._operation_administrators.php]

```
<?php
// Seguridad
include_once('acceso.php');
// Establecemos conexión con la base de datos
include_once('config.php');
```

```
// Función auxiliar para a generar XML
function send_xml($operation,$status) {
	// Generem XML
	$xmldata  = "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
	$xmldata .= "<UpdateStatus>\n";
	$xmldata .= "<Operation>$operation</Operation>\n";
	$xmldata .= "<Status>".($status ? "1" : "0")."</Status>\n";
	$xmldata .= "</UpdateStatus>\n";

	// Retornem el resultat
	header('Content-type: text/xml');
	header('Pragma: public');
	header('Cache-control: private');
	header('Expires: -1');
	echo($xmldata);

	// Finalitzem script
	die();
}

// Recuperamos parámetros
if(isset($_GET["operation"])){
	$operation = $_GET["operation"];
} else $operation = 'nula';

if($operation != 'nula'){
	if($operation == 'nuevo'){
		if(isset($_GET["newUser"])){
			$newUser = $_GET["newUser"];
		} else send_xml($operation,false);
		if(isset($_GET["pass"])){
			$pass = $_GET["pass"];
		} else send_xml($operation,false);
		if(isset($_GET["name"])){
			$name = $_GET["name"];
		} else send_xml($operation,false);
		if(isset($_GET["type"])){
			$type = $_GET["type"];
		} else send_xml($operation,false);
		$sql = "INSERT INTO
administrators(IdUser,Password,Name,IdAccessType)
VALUES('$newUser','$pass','$name',$type);";
		$result = mysql_query($sql);
	}
	if($operation == 'editar'){
		if ((!isset($_GET["IdUser"]))) {
			send_xml($operation,false);
		} else {
			$IdUser = $_GET["IdUser"];
		}
		if(isset($_GET["newUser"])){
			$newUser = $_GET["newUser"];
		} else send_xml($operation,false);
		if(isset($_GET["pass"])){
			$pass = $_GET["pass"];
		} else send_xml($operation,false);
		if(isset($_GET["name"])){
			$name = $_GET["name"];
			//send_xml($operation,false);
		} else send_xml($operation,false);
```

```
                if(isset($_GET["type"])){
                        $type = $_GET["type"];
                } else send_xml($operation,false);
                $sql = "UPDATE administrators SET IdUser = '$newUser',
Password = '$pass', Name = '$name', IdAccessType = $type WHERE IdUser =
'$IdUser';";
                $result = mysql_query($sql);
        }
        if($operation == 'borrar'){
                if ((!isset($_GET["IdUser"])) {
                        send_xml($operation,false);
                } else {
                        $IdUser = $_GET["IdUser"];
                }
                $sql = "DELETE FROM administrators WHERE IdUser =
'$IdUser';";
                $result = mysql_query($sql);
        }
}else{
        send_xml($operation,false);
}

// Generamos el resultado
send_xml($operation,$result);
?>
```

En esta página recuperamos el valor $operation, que podrá ser nuevo, editar o borrar, los valores nuevo o editar son enviados a través del formulario de edición e indican o bien que se trata de una edición de un administrador o bien una inserción de uno nuevo, en cualquiera de los dos casos se hace un INSERT o UPDATE (según el caso) de la base de datos y obtenemos el resultado en $result, con la operación borrar ocurre lo mismo, además de la operación necesitamos conocer el IdUser del administrador para poder borrarlo correctamente, si recordamos, la función borrarAdmin() de administrators.js es la que se encargaba de enviar estos valores a esta página, si algo de esto falla en $result tendremos un valor de false o 0 que será lo que devolvemos, al generar el resultado en forma de XML, tenemos la operación realizada y si se ha realizado o no correctamente, que una vez más es lo que utiliza la función borrarAdmin() para mostrarnos un resultado.

11.3. CSS

Para explicar la parte de estilos con CSS, usaremos como ejemplo el menú de navegación principal de nuestra página en el cual se puede observar la potencia que puede llegar a alcanzar las hojas de estilo.

A continuación podemos observar el código del mencionado menú, consiste en una cabecera <h1> con el título y luego una lista <ul> de elementos <li> que contienen enlaces <a> a las distintas páginas de nuestro módulo de administración, la lista <ul> está identificada por “mainNav”.

[admin_menu.php]

```
<h1>Servifot - Módulo de Administración</h1>
<ul id="mainNav">
	<li><a href="index.php">Inicio</a></li>
	<li><a href="admin_orders.php">Pedidos</a></li>
	<li><a href="admin_categories.php">Categorías</a></li>
	<li><a href="admin_products.php">Productos</a></li>
	<li><a href="admin_users.php">Usuarios</a></li>
	<li><a href="admin_administrators.php">Administradores</a></li>
	<li><a href="admin_tables.php">Tablas</a>
	<li class="logout"><a href="acceso.php?close=true">Desconectar</a></li>
</ul>
```

La hoja de estilo que usaría este menú sería el siguiente fragmento de código:

CSS para el menú de navegación

```
/* Menú principal */
#mainNav {
	width: 918px;
	height: 37px;
	padding: 6px;
	margin: 0 0 30px 0;
	background: url(../images/top-menu-bg.gif) no-repeat left top;
	list-style: none;
}
#mainNav li {
	float: left;
	height: 37px;
	border-right: 1px solid #ddd;
}
#mainNav li a {
	display: block;
	float: left;
	height: 37px;
	font-weight: bold;
	line-height: 37px;
	text-decoration: none;
	color: #646464;
	padding: 0 20px;
}
#mainNav li.logout {
	float: right;
	border-left: 1px solid #ddd;
	border-right: none;
}
#mainNav li a:hover {
	color: white;
	background: #f28E00;
}
```

La forma de utilizar CSS es muy sencilla, simplemente es darle una serie de propiedades (par propiedad – valor) a los elementos de la página que deben de estar identificados o bien deben de ser etiquetas básicas de HTML, así por ejemplo podemos observar que para el identificador #mainNav, usamos entre otras las propiedades *width* y *height* que sirven respectivamente para darle una

anchura y altura al campo identificado por ese nombre (en nuestro caso la lista), luego con *list-style:none* haríamos que desapareciesen los puntos iniciales de cada elemento de una lista, etc.

Siguiendo con otro ejemplo, con #mainNav li, estaríamos haciendo que los elementos <li> que pertenecen a mainNav tuvieran las propiedades especificadas, destacamos *float:left* que es una propiedad muy usada ya que hace que los elementos "floten" a la izquierda, haciendo así que se pongan uno al lado de otro haciendo que la lista pase a ser un listado horizontal de elementos.

La forma de usar una hoja de estilo en una de nuestras páginas es añadiendo una línea de código en la etiqueta <head> del documento, con el tipo de dato que estamos usando y su ruta, quedaría de la siguiente forma:

```
<head>
        <meta http-equiv="Content-Type" content="text/html;
        charset=utf-8">
        <title>Autentificación</title>
        <link rel=stylesheet href="css/admin.css" type="text/css" />
 </head>
```

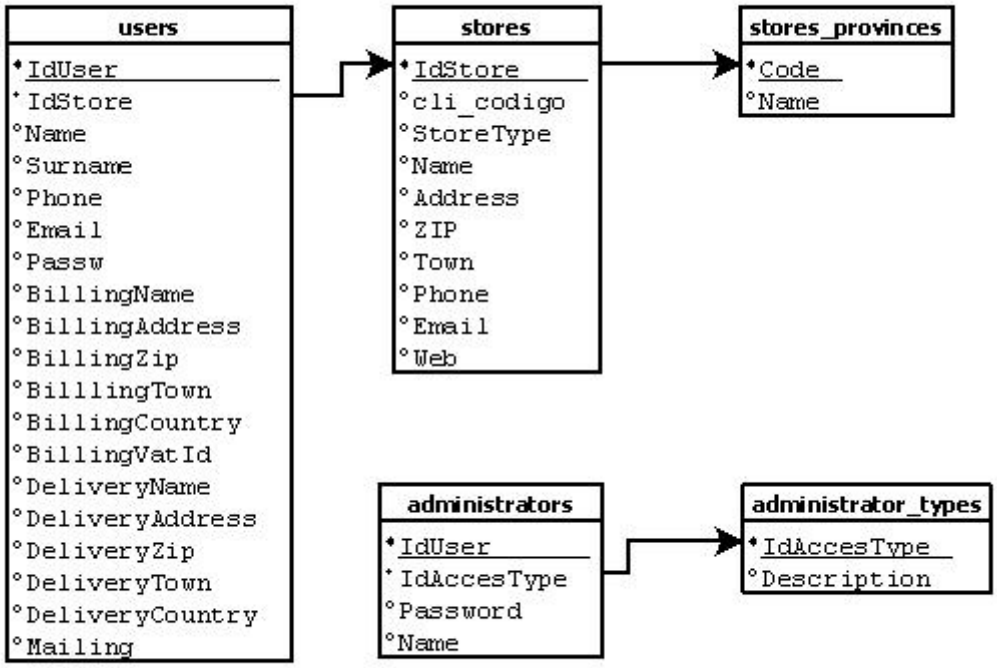
12. Estructura de la base de datos

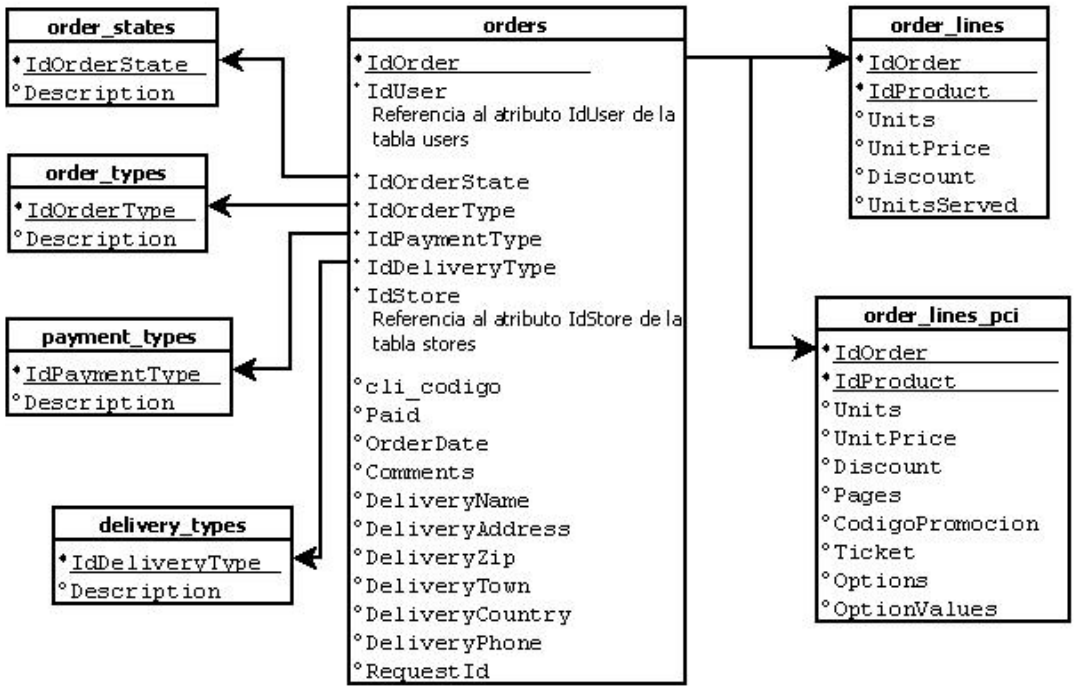
A continuación mostramos todo el diseño de la estructura de la base de datos de Servifot Store.

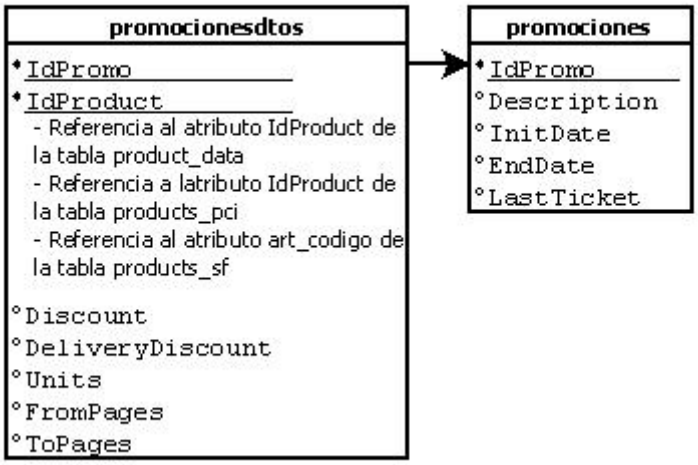
Los campos con un rombo grande en negrita y subrayados son la clave primaria de la tabla.

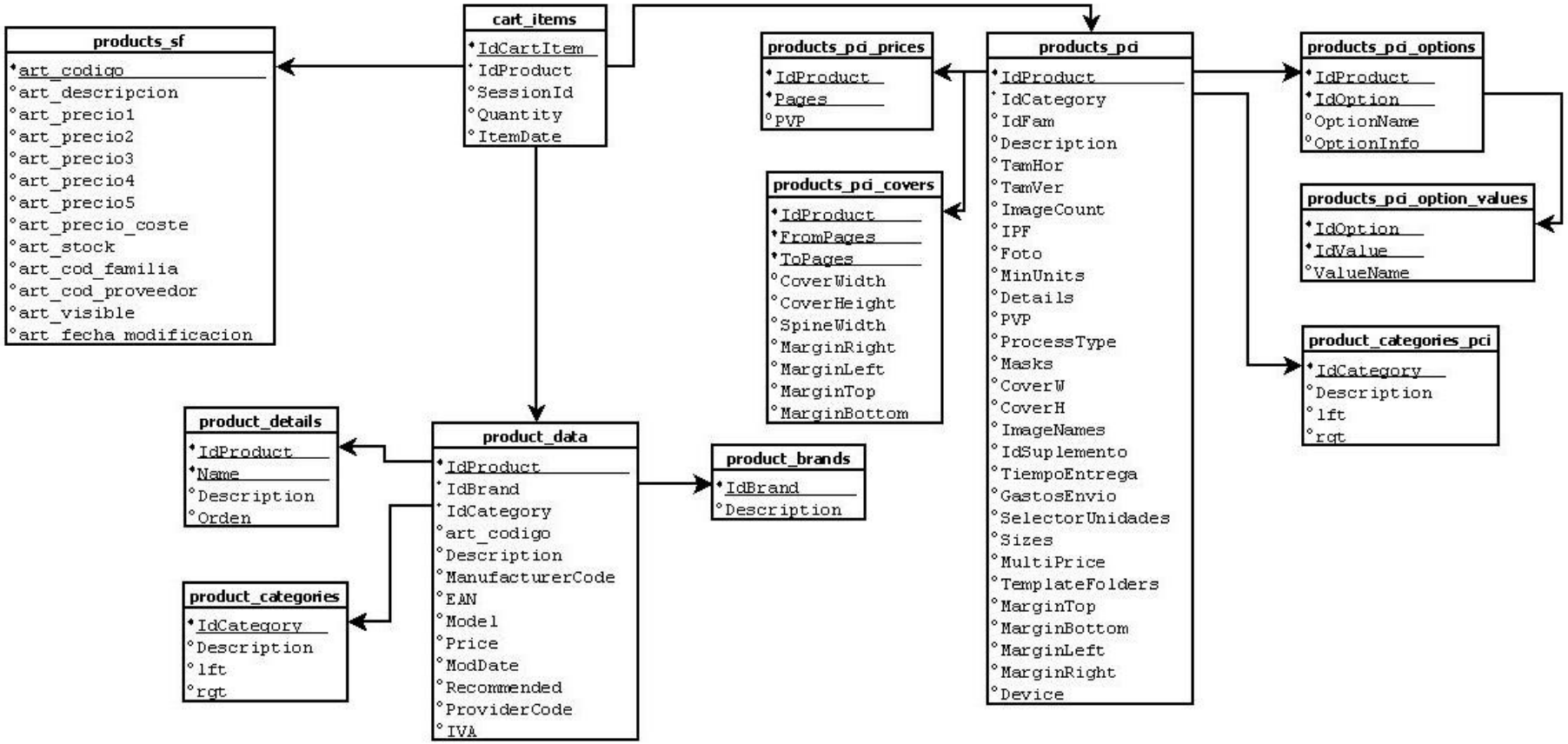
Los campos con un rombo más pequeño y sin subrayar son claves ajenas que hacen referencia a claves primarias de otras tablas.

Los demás campos (señalados con un circulito sin colorear) son atributos normales de la tabla.


users
IdUser
IdStore
Name
Surname
Phone
Email
Passw
BillingName
BillingAddress
BillingZip
BilllingTown
BillingCountry
BillingVatId
DeliveryName
DeliveryAddress
DeliveryZip
DeliveryTown
DeliveryCountry
Mailing
stores
IdStore
cli_codigo
StoreType
Name
Address
ZIP
Town
Phone
Email
Web
stores_provinces
Code
Name
administrators
IdUser
IdAccesType
Password
Name
administrator_types
IdAccesType
Description


order_states
IdOrderState
Description
order_types
IdOrderType
Description
payment_types
IdPaymentType
Description
delivery_types
IdDeliveryType
Description
orders
IdOrder
IdUser
Referencia al atributo IdUser de la tabla users
IdOrderState
IdOrderType
IdPaymentType
IdDeliveryType
IdStore
Referencia al atributo IdStore de la tabla stores
cli_codigo
Paid
OrderDate
Comments
DeliveryName
DeliveryAddress
DeliveryZip
DeliveryTown
DeliveryCountry
DeliveryPhone
RequestId
order_lines
IdOrder
IdProduct
Units
UnitPrice
Discount
UnitsServed
order_lines_pci
IdOrder
IdProduct
Units
UnitPrice
Discount
Pages
CodigoPromocion
Ticket
Options
OptionValues


promocionesdtos
IdPromo
IdProduct
- Referencia al atributo IdProduct de la tabla product_data
- Referencia a latributo IdProduct de la tabla products_pci
- Referencia al atributo art_codigo de la tabla products_sf
Discount
DeliveryDiscount
Units
FromPages
ToPages
promociones
IdPromo
Description
InitDate
EndDate
LastTicket


products_sf
art_codigo
art_descripcion
art_precio1
art_precio2
art_precio3
art_precio4
art_precio5
art_precio_coste
art_stock
art_cod_familia
art_cod_proveedor
art_visible
art_fecha_modificacion
cart_items
IdCartItem
IdProduct
SessionId
Quantity
ItemDate
products_pci_prices
IdProduct
Pages
PVP
products_pci
IdProduct
IdCategory
IdFam
Description
TamHor
TamVer
ImageCount
IPF
Foto
MinUnits
Details
PVP
ProcessType
Masks
CoverW
CoverH
ImageNames
IdSuplemento
TiempoEntrega
GastosEnvio
SelectorUnidades
Sizes
MultiPrice
TemplateFolders
MarginTop
MarginBottom
MarginLeft
MarginRight
Device
products_pci_options
IdProduct
IdOption
OptionName
OptionInfo
products_pci_option_values
IdOption
IdValue
ValueName
products_pci_covers
IdProduct
FromPages
ToPages
CoverWidth
CoverHeight
SpineWidth
MarginRight
MarginLeft
MarginTop
MarginBottom
product_categories_pci
IdCategory
Description
lft
rgt
product_details
IdProduct
Name
Description
Orden
product_data
IdProduct
IdBrand
IdCategory
art_codigo
Description
ManufacturerCode
EAN
Model
Price
ModDate
Recommended
ProviderCode
IVA
product_brands
IdBrand
Description
product_categories
IdCategory
Description
lft
rgt

13. Plugins Jquery

Los plugins de Jquery, son segmentos de código JavaScript que usan la librería Jquery y que pueden incluirse en las páginas webs de forma muy sencilla, existen infinidad de estos plugins, cada plugin realiza una función específica. Nosotros hemos usado dos de estos plugins, uno de ellos denominado Validate y el otro Datepick.

13.1. Validate

El plugin Validate se encarga de validar los campos introducidos en un formulario justo antes de realizar el envío de los datos del formulario, informando al usuario de posibles errores que puedan contener los datos introducidos y dando posibilidad a cambiarlos antes de volver a intentar enviarlos.

En la figura 50 podemos observar que ocurre cuando intentamos introducir en el módulo de administración un usuario con todos los campos en blanco.

Figura 50: Control de campos mediante el plugin Validate

La forma de utilizar el plugin es la siguiente:

```
function validarUser(){
	var valido = $("#admin_users").validate({
		rules: {
			name: {
				required: true
			},
			surname: {
				required: true
			},
			email: {
				email: true
			},
			pass1: {
				required: true,
				minlength: 6
			},
			pass2: {
				required: true,
				minlength: 6,
				equalTo: "#pass1"
			},
			nombre_fac: {
				required: true
			},
			direccion_fac: {
				required: true
			},
			postal_fac: {
				required: true
			},
			poblacion_fac: {
				required: true
			},
			pais_fac: {
				required: true
			},
		},
		messages: {
			name: {
				required: "Por favor, escriba su nombre"
			},
			surname: {
				required: "Por favor, escriba sus apellidos"
			},
			email: {
				email: "Por favor, escriba un correo electrónico"
			},
			pass1: {
				required: "Por favor, escriba su contraseña",
				minlength: "Debe tener, al menos, 6 caracteres"
			},
			pass2: {
				required: "Por favor, repita su contraseña",
				minlength: "Debe tener, al menos, 6 caracteres",
				equalTo: "Por favor, escriba la misma contraseña
				que arriba"
```

```
                        },
                        nombre_fac: {
                                required: "Por favor, escriba su nombre"
                        },
                        direccion_fac: {
                                required: "Por favor, escriba su dirección"
                        },
                        postal_fac: {
                                required: "Por favor, escriba su código postal"
                        },
                        poblacion_fac: {
                                required: "Por favor, escriba su población"
                        },
                        pais_fac: {
                                required: "Por favor, escriba su país"
                        },
                },
                submitHandler: function(form){
                        enviarDatos();
                }
        });
        if(valido){
                $('#admin_users').submit();
        }
}
```

Con el selector que hace referencia a nuestro formulario, en nuestro caso, #admin_users, debemos realizar una llamada a .validate, luego el plugin consta de tres partes, las reglas (rules), los mensajes (messages) y la acción a realizar al enviar el formulario (submitHandler), si todas las reglas se cumplen, la variable que hemos asignado al selector será true y en ese caso se realiza el envío del formulario, si no, aparecen los mensajes para cada regla no cumplida y no se envía nada. Cada regla tiene su correspondiente mensaje, las reglas funcionan añadiendo el identificador de cada campo y asignándole un conjunto de reglas (required, minlength, etc.…) que serán las que se comprueben al realizar el envío del formulario. Si alguna falla, se muestra su mensaje en el formulario. Cuando todo es correcto y pasamos a hacer el envío del formulario, realizamos las acciones del apartado submitHandler, que en este caso será una llamada a la función enviarDatos().

Además el plugin funciona en tiempo real, de forma que si introducimos la contraseña, la información proporcionada cambia, advirtiéndonos que debemos introducir como mínimo 6 caracteres.

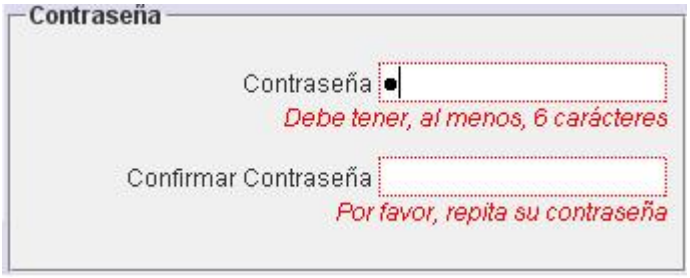


Figura 51: Plugin Validate para contraseñas (1)

Lo mismo ocurre cuando las contraseñas no coinciden, inmediatamente se nos informa.

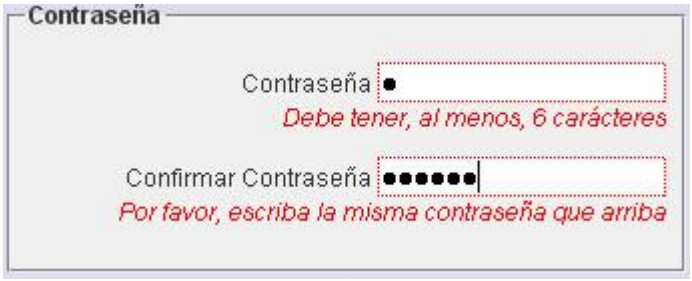


Figura 52: Plugin Validate para contraseñas (2)

13.2. Datepick

El plugin Datepick nos ayuda con la introducción de fechas, cuando pulsamos por ejemplo en el campo Fecha Inicio o Fecha Fin del formulario de edición de promociones, se despliega un calendario con los días para que simplemente pulsando sobre un día se introduzca en el recuadro correspondiente y con el formato necesario dicho día.

Figura 53: Plugin Datepick

La forma de usar este plugin es la siguiente:

```
$(function() {
	$("#initdate").datepicker({
		dateFormat: 'yy-mm-dd'
	});
	$("#enddate").datepicker({
		dateFormat: 'yy-mm-dd'
	});
});
```

Debemos seleccionar el selector que queremos que haga referencia al campo que contendrá la fecha, entonces hacemos una llamada a .datepicker para cada uno de esos selectores, en nuestro caso son dos, la fecha de inicio y la fecha de fin, además de eso, con el atributo dateFormat podemos decirle que formato queremos que tenga la fecha que seleccionemos en el calendario.

14. Software utilizado

A lo largo de todo el proyecto se han usado distintas aplicaciones y software para la confección del mismo. Pasemos a comentar cada uno de ellos de forma breve.

En primer lugar, el más utilizado de todos ha sido la aplicación Aptana, funciona como plugin de Eclipse pero puede funcionar de forma independiente, con este programa hemos podido tratar todo el código tanto PHP, JS, CSS, HTML, etc.

Figura 54: Logotipo Aptana

Para comprobar nuestros progresos hemos usado dos de los navegadores más usados en PC durante la última década, Mozilla Firefox e Internet Explorer. En alguna ocasión hemos probado Google Chrome, de reciente aparición y que ofrecerá mucho juego en un corto espacio de tiempo seguramente.

Figuras 55 y 56: Logotipo Firefox y Logotipo IE8

Para subir nuestros ficheros al servidor hemos usado un plugin de Firefox denominado FireFTP.

Figura 57: Logotipo FireFTP

También hemos usado otro plugin de Firefox, denominado FireBug para realizar una depuración de las llamadas AJAX y del CSS de los documentos de nuestras páginas. Este plugin es muy útil para observar la estructura del documento y poder trabajar con CSS más cómodamente.

Figura 58: Logotipo FireBug

A la hora de crear los botones y demás imágenes de nuestras páginas hemos usado el conocido PhotoShop.

Figura 59: Logotipo Photoshop

Para realizar el diseño del esquema de la estructura de datos de la base de datos en una primera aproximación antes de tratar directamente con la propia base de datos hemos usado la aplicación DIA.

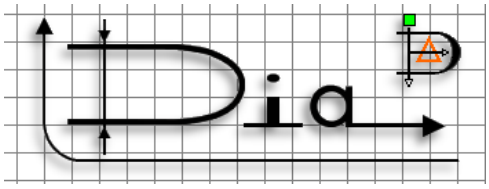


Figura 60: Logotipo DIA

Y finalmente para tratar con la base de datos directamente hemos usado MySQL Query Browser, que nos da acceso fácilmente a la base de datos y nos permite manipularla a nuestro antojo.

Figura 61: Logotipo MySQL Query Browser

15. Bibliografía

Bear Bibeault y Yehuda Katz. *Jquery in Action*. Manning, 2008.

Jonathan Chaffer y Karl Swedberg. *Learning Jquery*. Packt, 2007.

Jonathan Chaffer y Karl Swedberg. *Jquery Reference Guide*. Packt, 2007.

Anthony T. Holdener. *Ajax: The Definitive Guide*. O'Reilly, 2008.

Christian Gross. *Ajax Patterns and Best Practices*. Apress, 2006.

Kris Hadlock. *Ajax for Web Application Developers*. Sams, 2006.

Richard York. *CSS Instant Results*. Wrox Press, 2006.

Elizabeth Castro. *HTML, XHTML, & CSS*. Peachpit Press, 2006.

Paul Haine. *HTML Mastery: Semantics, Standards and Styling*. Apress, 2006.

Danny Goodman y Michael Morrison. *Javascript Bible, 5th Edition*. Wiley, 2004.

David Sawyer McFarland. *CSS: The Missing Manual*. O'Reilly, 2006.

Mike Hillyer. *Managing Hieriarchical Data in MySQL*. Disponible en:
http://dev.mysql.com/tech-resources/articles/hierarchical-data.html

Manual de PHP. Disponible en:
http://php.net/manual/es/index.php

Liam Quin. *Extensible Markup Language (XML)*. Disponible en:
http://www.w3.org/XML/

MySQL 5.0 Reference Manual. Disponible en:
http://dev.mysql.com/doc/refman/5.0/es/index.html

The JQuery Project. Disponible en:
http://jquery.com/