SISTEMA GENÉRICO PARA LA UTILIZACIÓN DE DATOS ESTRUCTURADOS E HIPERVICULADOS

TESIS

Que para obtener el Título de
INGENIERO ELECTRICISTA

Presenta
Francisco Tzintzun Márquez

Asesor de Tesis
M.C. Luis Rubén Rusiles Zamora

Universidad Michoacana de San Nicolás de Hidalgo
Julio del 2007

Agradecimientos

Esta tesis debe mucho, en su producción, a la dedicación y esfuerzo del M.C. Luis Rubén Rusiles Zamora por su ayuda y estimulantes comentarios sobre la mayor parte de los capítulos, así como en la elección del título de proyecto.

Agradezco también a los miembros de la mesa sinodal por sus observaciones e indicaciones hechas a mi trabajo.

Debo agradecer también a la Facultad de Ingeniería Eléctrica de la Universidad Michoacana de San Nicolás de Hidalgo por todos estos años de formación profesional en mis estudios de nivel licenciatura.

Dedicatoria

Con amor, respeto y cariño a mis padres:

Ramón Tzintzun y Juana Márquez, quienes desde pequeño me han introducido en un mundo de comprensión, análisis y aplicación.

Resumen

Actualmente el disponer de cualquier tipo de información perteneciente a un grupo, asociación o negocio, de forma organizada, clara y consultable a través de Internet es una parte de nuestra vida cotidiana. Sin embargo, algunas de las herramientas de almacenamiento de información tienen en común el no ser útiles a personas que desconocen, los métodos y lenguajes de programación. Para superar esta desventaja se construyó un sistema de gestión de bases de datos que permite construir bases de datos basadas en la descripción semántica de objetos, proporcionando así una mejor familiarización del usuario con el programa. El sistema cuenta con interfaces mediante formularios para la administración de usuarios y permisos, la creación de bases de datos y la construcción de clases a través de la implementación de tablas. La información de cada clase puede relacionarse con otras clases así como agruparse y heredar las características de una clase a otra. La manipulación y acceso de datos es realizada por medio del servidor Apache, por lo que también es accesible a través de Internet. Este documento tiene como objetivo exponer el planteamiento de construcción, las características de diseño y el funcionamiento del sistema desarrollado.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Símbolos y Abreviaturas

BDOO	Bases de datos orientadas a objetos.
CGI	*Common Gateway Interface*. Clase de interfaz común de pasarela.
CSS	*Cascade Style Sheets*. Hojas de estilo en cascada.
DBI	*Database Independent Interface*. Interfaz independiente de base de datos.
DCL	*Data Control Language*. Lenguaje de control de datos.
DDL	*Data Definition Language*. Lenguaje de definición de datos.
DML	*Data Management Language*. Lenguaje de manipulación de datos.
GPL	*General Public License*. Licencia publica general.
HTML	*Hypertext Markup Language*. Lenguaje de marcado de hipertexto.
PERL	*Practical Extraction and Report Language*. Lenguaje practico de extracción e informe.
POO	Programación Orientada a Objetos.
SGUDEH	Sistema genérico para la utilización de datos estructurados e hipervinculados.
SO	Sistema operativo.
URL	*Uniform Resource Locators*. Localizador uniforme de recursos.

Capítulo 1

Introducción

"Esto es algo que exige una total consideración, con mil días de práctica para el entrenamiento y diez mil días de práctica para refinarlo."
Musashi Miyamoto – El libro de los cinco anillos

"Porque, ¿quien de vosotros, queriendo edificar una torre, no se sienta primero y calcula los gastos, a ver si tiene lo que necesita para acabarla?"
S. Lucas 14:31

Capítulo 1

Introducción

El presente documento trata el planteamiento, diseño e implementación de un sistema de gestión de bases de datos orientadas a objetos, denominado "Sistema Genérico para la Utilización de Datos Estructurados e Hipervinculados" (SGUDEH). Este sistema permite resolver diversos problemas, tales como los de desarrollo de sistemas de almacenamiento de información, sin necesidad de tener conocimientos de un lenguaje de programación, ni de un sistema de gestión de datos. La administración y la seguridad de datos es realizada mediante un servidor *Web*, lo cual permite que los datos sean consultados de forma segura por *Internet*.

En la mayoría de los sistemas de gestión de datos disponibles en la actualidad, es necesario estar familiarizado con algún lenguaje de programación para crear tablas, agregar información, consultar información y relacionarla con la información de otras tablas. En otros programas como *Excel*, además de tenerse costos muy elevados de sus licencias, se requiere trabajar manualmente con las tablas y sus relaciones, lo cual no es fácil.

Por lo tanto, para dar una solución a estos problemas, se plantea un sistema desarrollado con un modelo sencillo para la implementación de tablas y actualización de información, empleando herramientas disponibles bajo la licencia *GNU GPL* (*General Public License*), lo cual produce un bajo costo de la aplicación.

En este capítulo se realiza una presentación breve de los antecedentes sobre bases de datos orientadas a objetos. Se hace la propuesta de un problema de ejemplo a usar como base para el desarrollo del proyecto, el cual fue resuelto utilizando el Sistema Genérico para la Utilización de Datos Estructurados e Hipervinculados (SGUDEH). Al final del capítulo se presenta de forma breve el contenido del resto de los capítulos que componen la tesis.

1.1 Antecedentes

El uso de sistemas de bases de datos automatizadas se desarrolla a partir de la necesidad de almacenar gran cantidad de información y de realizar consultas. Se dice que los sistemas de bases de datos tienen su origen durante la década de 1950 gracias al proyecto estadounidense Apollo, el cual pretendía enviar el primer hombre a la luna. En esta época no existía ningún sistema que permitiera gestionar la inmensa cantidad de información que requería el proyecto [BUAP 2007].

En la década de 1960, se inicia la primera generación de bases de datos, la cual consistía en sistemas de ficheros, predominando los dos modelos siguientes:

1. El modelo de datos jerárquico, que estaba basado en relaciones de árbol (padres e hijos). Un ejemplo de este modelo es el sistema *IMS* (*Information Management System* 1968) desarrollado por *IBM* a partir del sistema utilizado en el programa Apollo de la *NASA*.
2. El modelo de datos en red, mejor conocido como *CODASYL* (*Conference Ondata Systems Language*), estaba basado en un grafo de relaciones entre datos.

En la década de 1970, surgieron los modelos de segunda generación. Entonces se inicia el modelo de datos relacional desarrollado por Edgar F. Codd, en el cual los datos son organizados como tablas relacionadas, este tipo de modelo es el que actualmente domina el mercado de bases de datos. Durante esta década se generan dos grandes desarrollos:

- *SQL* (*Structured Query Language*), desarrollado por *IBM*, es actualmente un estándar para la utilización de bases de datos.
- *ORACLE*, desarrollado por *ORACLE CORPORATION*, el cual recientemente ha implementado la tecnología orientada a objetos.

Surgen también los sistemas relacionales para microprocesadores como *dbaseIV* desarrollado por *Borland* y *Access* desarrollado por *Microsoft* para computadoras personales.

A partir de la década de 1990 se inicia el surgimiento de los sistemas de tercera generación, entre los cuales se encuentran:

- Las bases de datos distribuidas que implementan el almacenamiento de datos en múltiples localizaciones físicas.

- El modelo de datos orientado a objetos, que permiten implementar fácilmente relaciones de datos mucho más complejas, que actualmente son objeto de investigación y estudio, siendo sus aplicaciones usadas en distintas áreas, desde ingeniería hasta multimedia.

Una de estas investigaciones es en la cual el presente proyecto está basado, "Una Aproximación Axiomática A Las Bases De Datos Deductivas" [Antunes 1995], donde se presenta un análisis matemático para el modelado de bases de datos orientadas a objetos y se propone un método deductivo para la implementación de bases de datos orientadas a objetos.

1.2 Objetivo

El objetivo de esta tesis radica en realizar una aplicación tipo *Web*, mediante software libre y de licencia pública *GNU GPL* [GNU 2007]. Que al mismo tiempo facilite la implementación de bases de datos orientadas a objetos y el modelado de los datos mediante las relaciones: inclusión, atribución, composición por tupla y composición por conjunto. Por lo cual, esta herramienta estará orientada al usuario no programador para permitirle el diseño, almacenamiento y la modificación intuitiva de los datos. También para ofrecerle un sistema multiplataforma, que sea ejecutable en dos sistemas operativos: *Windows* y *Linux*.

1.3 Justificación

El campo de las bases de datos orientadas a objetos se ha introducido como una nueva área de investigación de la cual han derivado investigaciones y proyectos. Sin embargo en dichos proyectos, en su mayoría, es necesario tener conocimientos de su lenguaje de programación y las investigaciones resultan ser sumamente difíciles.

Por lo anterior, se desea implementar un programa en el cual no sea necesario el conocimiento de lenguajes y/o estándares de programación y que permita que el desarrollo de bases de datos se de, en una forma fácil e intuitiva, resultando en un sistema que implemente un lenguaje familiar con el usuario y que proporcione relaciones entre objetos definidos. De la misma manera, debe ser posible la disposición de la información a través de Internet y que el uso del programa no esta limitado a una sola plataforma de sistema operativo.

1.4 Metodología

La metodología usada consistió en el uso de un modelo orientado a objetos, sobre una base de datos relacional, que produce páginas dinámicas. Para la documentación se uso un problema base, el cual es referido a lo largo del proyecto para establecer los conceptos de implementación de relaciones de clases y fue implementado mediante el sistema genérico (SGUDEH) construido. Esta implementación se describe en el capítulo 5. El problema propuesto cubre todas las relaciones de clases que se han propuesto de forma clara.

El problema de ejemplo es el siguiente: "Una compañía cuya actividad principal es la venta de inmuebles desea construir una base de datos, la cual le permita tener control de:

- El control de ventas de las obras y de clientes.
- La obras en existencia
- La relación de obras realizadas por constructor.

Teniendo los siguientes inmuebles en venta con el costo por unidad de construcción y de venta, mostrados en la tabla 1.1.

Tabla 1.1. Problema de ejemplo: relación de obras existentes.

	Casas		Departamentos		Locales	
Estilo	Zafiro	Diamante	Rústico	Mediterráneo	Chico	Grande
Cantidad	10	20	10	10	10	5
Costo por construcción	$136,000	$178,500	$119,000	$170,000	$51,000	$68,000
Costo de venta	$160,000	$210,000	$140,000	$200,000	$60,000	$80,000

De modo que las obras construidas y en construcción fueron distribuidas entre los constructores de la inmobiliaria como se muestra en la tabla 1.2.

Tabla 1.2. Problema de ejemplo: distribución de obras por constructor.

	Casas		Departamentos		Locales	
	Zafiro	Diamante	Rústica	Mediterráneo	Chicos	Grandes
Constructor 1		20		10		5
Constructor 2	10		5			
Constructor 3			5		10	

Se deben presentar las siguientes distribuciones de información:

1. Se debe tener un catálogo de las obras vendidas por cliente.
2. Debe contar con tres listas: departamentos por edificios, casas por fraccionamientos y locales por centros comerciales.
3. Cada expediente es compuesto por toda la documentación de las obras asignadas a un constructor.
4. La documentación esta compuesta por los distintos tipos de obras asignadas a un constructor para ser construidas.
5. Un cliente puede comprar y apartar una o varias obras, y cada obra es vendida a un cliente.

1.5 Contenido de la tesis

En el capítulo 1 se presentan algunos datos relevantes de la investigación previa realizada referente a las bases de datos deductivas orientadas a objetos. Se indica también la forma en que se abordará la elaboración del sistema genérico, usando como referencia un problema propuesto.

En el capítulo 2 se presentan los lenguajes de programación utilizados en la elaboración del proyecto. Se hace referencia a los conceptos más utilizados de cada lenguaje.

En el capítulo 3 se presenta la teoría de relaciones entre objetos y la forma en que se traducen estas relaciones a una base de datos relacional.

En el capítulo 4 se presenta la forma en que se implementa el sistema utilizando el punto de vista de las bases de datos relacionales.

En el capítulo 5 se presenta la resolución del problema base mediante el sistema genérico.

En el capítulo 6 se presenta la documentación correspondiente al sistema genérico.

En el capítulo 7 se presentan conclusiones generales y personales así como ideas para posibles trabajos futuros, o extensiones del sistema construido.

En el apéndice A se presentan las directivas básicas de Apache 2.0, utilizadas para la configuración de los archivos de servidor, así como para los archivos de contraseña.

En el apéndice B se presentan los permisos de *MySQL* y que facultades otorgan a los usuarios.

En el apéndice C se presentan los archivos que componen el sistema genérico y las funciones que realiza cada archivo.

Capítulo 2

Lenguajes de Programación

"Aún entre personas que hablan el mismo idioma siempre es necesario hacer una interpretación del modo de pensamiento"
Takasi Matsuoka - El honor del samurai

"*Los pensamientos elevados requieren un lenguaje elevado*"
Aristófanes

Capítulo 2

Lenguajes de Programación

En el presente capítulo se hace una breve introducción a los programas usados para la implementación del proyecto (SGUDEH). Estos programas son conocidos como *LAMP* (*Linux*, Apache*, MySQL, Perl*) [LAMP 2007]:

1. *Linux* es un sistema operativo.
2. Apache es un servidor *Web*.
3. *MySQL* es un sistema de gestión de bases de datos.
4. *Perl* es un lenguaje de propósito general, principalmente enfocado al desarrollo de aplicaciones tipo *Web*.

El uso de estos programas esta ampliamente difundido para desarrollar servidores de información porque son aplicaciones de código abierto (*Open Source*) y con licencia pública *GNU GPL*.

Se hace énfasis en tres puntos importantes:

- La configuración de Apache para archivos *CGI*.
- La sintaxis más usual para las principales sentencias de consulta y modificación de tablas de *MySQL*.
- Los módulos *CGI* y *DBI* de *Perl*.

Se incluyen también algunas notas básicas de *HTML*. Señalando finalmente que todos estos programas tienen una muy extensa y completa documentación en sus sitios oficiales en *Internet*.

2.1 Linux

Es un sistema operativo basado en software libre, el cual no comparte origen ni diseño con ningún *Unix*. Fue inicialmente creado por Linus Torvalds en 1991, como proyecto de tesis. Se debe aclarar que *Linux* es sólo el *kernel* y *GNU* es el conjunto de aplicaciones. Las distribuciones existentes son recopilaciones de un *kernel* de *Linux* y las aplicaciones

seleccionadas. Existe un conjunto de distribuciones disponibles basadas en el kernel de *Linux*: *Debian*, *Red-Hat* (*Fedora*), *Mandriva*, *Suse*, *Ubuntu*, etc.

2.2 Apache

Apache surgió a partir del servidor *HTTPd* de *NCSA* (*Nacional Center for Supercomputing Applications*) versión 1.3 en 1995 [Apache 2007]. Actualmente, es uno de los mejores servidores de *HTTP* capaz de competir con otros servidores *HTTP* basados en *Unix* en cuanto a flexibilidad, velocidad y rendimiento. Apache es usado en más del 68% de todos los servidores *Web*[1], además es altamente configurable y extensible a través de módulos [Sromero 2006]. Incluye la posibilidad de ofrecer servicios de *Internet* e *Intranet* a cualquier usuario del propio sistema, proporcionando una dirección propia del tipo: http://nombre_maquina/~usuario.

Actualmente Apache cuenta con versiones para los sistemas: *Windows*, *Netware*, *OS/2*, *Unix*, *Linux*, etc. Para obtener información acerca de la instalación de apache se encuentra disponible el sitio oficial de Apache (http://www.apache.org), o en la publicación electrónica Apache *Week* (http://www.apacheweek.com).

2.2.1 El protocolo CGI

El protocolo *CGI* (*Common Gateway Interface*) fue creado para establecer un protocolo estándar de comunicación entre el servidor y el cliente de *Web*, de forma tal que a través de cualquier lenguaje de programación puedan recibirse datos que el usuario envía usando el método *POST* (envía los datos a través de la entrada estándar) ó *GET* (envía los datos a través de la variable de entorno *QUERY_STRING)*. Típicamente para recibir datos se usa alguna biblioteca o módulo del lenguaje elegido que implementa el protocolo *CGI*, para enviar datos simplemente se envían a la salida estándar desde el lenguaje elegido y el servidor se encarga de redireccionar esto al navegador.

Dentro de las aplicaciones se usa algún mecanismo para recuperar los datos enviados por el navegador desde las variables de ambiente (todos los lenguajes manipulan variables de ambiente). El protocolo *CGI* justamente consiste en especificar de qué forma los datos enviados por el navegador se convierten en variables de ambiente, esto en general

[1] Datos tomados de encuesta realizada por Netcraft en Marzo del 2006

es transparente al usuario. De esta forma pueden realizarse aplicaciones para un sitio *Web* en casi cualquier lenguaje, siendo esta la razón por la cual los lenguajes interpretados rápidamente ganaron terreno ya que tienen un ciclo de desarrollo en tiempo inferior a los lenguajes compilados y son más fáciles de depurar dentro del ambiente *CGI*.

2.2.2 Estructura de los archivos de configuración de Apache

Los archivos de Apache normalmente se instalan en las carpetas mostradas en la tabla 2.1, aunque la localización puede variar en los diferentes sistemas.

Tabla 2.1 Carpetas del servidor Apache.

	Linux	*Windows*
archivos binarios	/usr/local/bin o en /usr/sbin	c:\apache\bin
ficheros de configuración	/etc/httpd/conf	c:\apache\conf
logs	/etc/httpd/logs	c:\apache\logs
manuales o documentación	/home/httpd/html/manual	c:\apache\html\manual

A partir de la versión 1.3.4 del servidor Apache sólo se utilizaba un único fichero de configuración (httpd.conf), para evitar posibles redundancias o confusiones. Las versiones anteriores (hasta la 1.3.3) conservaban tres ficheros de configuración, mostrados en la tabla 2.2

Tabla 2.2 Ficheros de configuración de Apache.

httpd.conf	Datos de control del servidor.
access.conf	Datos para el control de accesos.
Srm.conf	Datos sobre especificación de ficheros. localización de las páginas, el nombre de directorios de usuarios), etc.

A partir de la versión 2.0 de Apache, se utiliza también un directorio (normalmente conf.d) donde se localizan los ficheros de configuración complementarios para los módulos añadidos. Desde el programa de configuración pueden cargarse otros ficheros adicionales

(usando la orden *include*), para distribuir lógicamente las opciones relacionadas con los módulos extra o con características especiales.

El fichero *httpd.conf*, y cualquier otro archivo llamado por éste, consta de una sintaxis basada en una serie de directivas de configuración, que se pueden clasificar en:

- **Simples**: una directiva por línea.
- **Compuestas**: bloque de código que incluye una o varias directivas, tanto simples como complejas.

Las líneas con comentarios comienzan con el símbolo #. Adicionalmente se distribuye otro fichero de configuración (mime.types) que permite especificar los tipos de documentos que serán suministrados por el *HTTPd*. La tabla mostrada en el apéndice A, muestra algunas de las directivas más comunes para configurar un servidor Apache 2.0.

El servidor Apache incluye también varias directivas que permiten restringir o verificar el acceso a determinados documentos. Las subdirectivas incluidas en las cláusulas <*Directory*>, <*Files*> y <*Location*> establecen los permisos básicos para acceder a directorios, ficheros y *URLs* específicos, respectivamente. La tabla en el apéndice B muestra las directivas más comunes permitidas.

La autentificación de usuarios y grupos se realiza a través de una serie de ficheros que deben estar fuera del árbol de directorios de documentos, y únicamente con permisos de lectura para el usuario ejecutor de los procesos del servidor (directiva *User*). El fichero típico para el control de las claves de los usuarios definidos para acceso al *Web* tiene el siguiente formato:

Usuario:Clave

La tabla mostrada en el apéndice C muestra las directivas de configuración que gestionan los ficheros de usuarios en Apache 2.0.

Una vez instalado e iniciado el servidor de Apache (*Apache Server Daemon*), se dirige un navegador (*Internet Explorer*, *Netscape, Lynx* ó similar) hacia el propio servidor de la máquina (*localhos*t, la dirección IP de la maquina o el nombre de máquina que tenga asociado). Si aparece la página de prueba de Apache (por ejemplo, dirigiendo *Netscape* a http://localhost, o a http://127.0.0.1) la instalación y configuración ha sido correcta y se puede proceder a configurar el uso de archivos *CGI* en el servidor *Web*.

2.2.3 Configuración para archivos CGI de Perl

Para configurar el uso de archivos *CGI* desarrollados en *Perl* con extensiones .cgi, .pl y .pm, se configura el archivo httpd.conf al cual se le deben incluir las siguientes directivas:

Paso 1. Se busca la línea cuyo enunciado inicia con la palabra ***Options***:

Options *Indexes FollowSymLinks*

Se le agrega la opción ***ExecCGI*** como se muestra a continuación:

Options *Indexes FollowSymLinks* ***ExecCGI***

Paso 2. Se busca la línea cuyo enunciado inicia con la frase ***AddHandler*** ó se agrega en caso de no existir*:*

AddHandler *cgi-script* .cgi

Se agrega las extensiones .pl, .pm y .cgi, como sigue:

AddHandler *cgi-script* **.cgi .pl .pm**

Paso 3. Para que los usuarios dispongan de un sitio propio en el servidor y se pueda publicar material en *Internet*, se utiliza una carpeta denominada ***public_html***, agregase lo siguiente:

```
<Directory /home/*/public_html>
        AllowOverride FileInfo AuthConfig Limit
        Options MultiViews Indexes SymLinksIfOwnerMatch ExecCGI
        <Limit GET POST OPTIONS>
                Order allow,deny
                Allow from all
        </Limit>
```

```
    <LimitExcept GET POST OPTIONS>
        Order deny,allow
        Deny from all
    </LimitExcept>
</Directory>
```

2.2.4 Variables de entorno

El servidor Apache pone a disposición de los *scripts* una serie de variables con información que puede resultar muy útil en determinadas ocasiones; son denominadas variables de entorno, y pueden ser leídas desde un programa en ejecución. Estas variables se asignan de forma automática en la lista asociativa *%ENV* en *Perl* y cada una tiene una clave que le identifica. Algunos ejemplos de estas variables son mostrados en la tabla 2.3. En el apéndice A se muestran todas las variables que aparecen en *Windows* y *Linux*.

Tabla 2.3 Variables de ambiente.

Clave	Valor
QUERY_STRING	Cadena de parámetros enviados por el método *GET*.
REQUEST_METHOD	Método empleado para el envió de parámetros (*GET* ó *POST*).
DOCUMENT_NAME	Nombre de la página que ha llamado al *script*.
CONTENT_LENGTH	Número de *bytes* de la cadena de parámetros.

2.3 HTML

HTML (*Hypertext Markup Language*) es un lenguaje para estructurar documentos a partir de texto en *World Wide Web*. Este lenguaje se basa en etiquetas (instrucciones que indican al texto como debe mostrarse) y atributos (parámetros que dan valor a la etiqueta), por lo que no describe la apariencia del diseño de un documento sino que permite a cada plataforma el darle un formato según su capacidad y la del navegador (tamaño de la pantalla, fuentes que tiene instaladas, etc..).

Básicamente, los documentos escritos en *HTML* constan del texto mismo del documento y las etiquetas que pueden llevar atributos, cualquier cosa que no sea una etiqueta es parte del documento mismo, esto vendría a ser:

<ETIQUETA> texto afectado </ETIQUETA>

La etiqueta del principio activa la orden y la última (que será idéntica a la del principio precedida del signo /) la desactiva, no todas las etiquetas tienen principio y final. Existen etiquetas que describen la estructura general de un documento y dan una información sencilla sobre él. Estas etiquetas no afectan a la apariencia del documento y solo interpretan y filtran los archivos *HTML*, mostradas en la tabla 2.4.

Tabla 2.4 Principales etiquetas descriptivas de documentos *HTML*.

<*HTML*>	Limitan el documento e indica que se encuentra escrito en este lenguaje.
<*HEAD*>	Especifica el prólogo del resto del archivo.
<*TITLE*>	Utilizado por los marcadores del navegador e identifica el contenido de la página. Solo puede haber un título por documento.
<*BODY*>	Encierra el contenido del documento.

2.3.1 Formato del párrafo

Las etiquetas de la tabla 2.5 modifican la estructura general de un documento y dan un formato al texto que se encuentra entre ellas.

Tabla 2.5 Principales etiquetas para formato de *HTML*.

<*H1*>, <*H2*>, ..., <*H6*>	Titulares: Sirven para dividir el texto en secciones. Se pueden definir seis niveles de titulares (desde <*H1*> hasta <*H6*>), el texto se coloca como sigue: <*H1*> Titular </ *H1*>.
<*P*>	Párrafos: Esta etiqueta se coloca al final de un texto indicando que a continuación se quiere una línea en blanco.
<*BR*>	Esta etiqueta sirve para realizar un salto de línea.

2.3.2 Creación de enlaces

Para generar un enlace a otro documento se necesita el nombre de un archivo o la dirección *URL* (es el Localizador Uniforme de Recursos, la dirección que localiza una información dentro de *Internet*) y el texto que servirá de punto de activación del otro documento (el cual

se mostrará en el documento y servirá como enlace). Los enlaces se generan mediante la etiqueta <*A*>texto</*A*>, llevará siempre dentro de la etiqueta un atributo ya sea:

<*A HREF*=""> o <*A NAME*="">

HREF es el atributo más habitual y sirve para saltar entre diferentes *URL*. Para saltar en una presentación del archivo 1 al archivo 2. En el archivo 1 se incluirá la directiva

<*A HREF*="archivo2.html">Siguiente página</*A*>

Si se conoce el nombre del campo de un formulario y se desea pasar cierta información mediante la entrada estándar se puede crear un enlace de la siguiente manera:

<*A HREF*="archivo2.html?campo1=dato1& campo2=dato2">Siguiente página</*A*>

2.3.3 Tablas

Las tablas permiten representar y ordenar cualquier elemento de una presentación en diferentes filas y columnas de modo que se puedan resumir grandes cantidades de información en una forma rápida y fácil. El contenido de una tabla se debe escribir entre las etiquetas:

<*TABLE*>...</*TABLE*>

Las tablas se definen fila a fila, celda a celda, comenzando desde la celda superior izquierda. Las etiquetas se definen como se indica en la tabla 2.6.

Tabla 2.6 Principales etiquetas para el uso de tablas de *HTML*.

<*TR*> …</*TR*>	Cada fila de la tabla se indica mediante las etiquetas
<*TH*> …</*TH*>	Indican las filas individuales dentro de cada fila. También indican que se trata de celdas que sirven como encabezado de tabla.
<*TD*> …</*TD*>	Indican las filas individuales dentro de cada fila. También indican que se trata de celdas comunes.

Las etiquetas de la tabla 2.7 modifican a las etiquetas de tabla, columna, fila y encabezado de tabla.

Tabla 2.7 Principales atributos para las tablas HTML

ALIGN	Define horizontalmente los datos al margen izquierdo (*left*), al derecho (*right*) o centrado (*center*).
VALIGN	Define verticalmente los datos en la parte superior (*top*), en la parte inferior (*bottom*) o centrado (*middle*).
ROWSPAN	Esta etiqueta más un valor, indica el número de filas que se abarcará.
COLSPAN	Esta etiqueta más un valor, indica el número de columnas que se quiere abarcar.
WIDTH	Acompaña a la etiqueta <*table*> y especifica el ancho de la tabla, tanto en número de píxeles como en porcentaje respecto al ancho de la pantalla. También puede acompañar a las etiquetas <*th*> ó <*td*> para especificar el ancho de las columnas.
BORDER	Acompaña a la etiqueta <*table*> Se le puede asignar un valor que indicará el ancho del borde en píxeles. *Border*="0" indica la ausencia de borde.
CELLPADDING	Acompaña a la etiqueta <*table*> indicando el espacio en píxeles entre el borde de la celda y su contenido. El valor predeterminado suele ser 1.

2.3.4 Formularios

Los formularios permiten, desde dentro de una presentación *Web*, solicitar información. Están compuestos por un número indefinido de campos y una vez introducidos los valores son enviados a una *URL* donde se procesará toda esta información. Como un formulario sigue siendo lenguaje *HTML* necesita de unas etiquetas que lo especifiquen. La declaración de formulario queda recogida por las etiquetas:

<form>…</form>

Dentro de ellas se recogerán todas las variables de entrada. A la etiqueta de apertura *<form>* se le asignan los atributos de la tabla 2.8.

Tabla 2.8 Principales atributos para la etiqueta *form*.

ACTION	Indica el programa que va a tratar las variables enviadas con el formulario, un guión *CGI* ó una *URL*.
METHOD	Indica el método de transferencia de las variables. *POST*, si se envía a través de la entrada estándar. *GET*, si se envía a través de la *URL*.

Debido a que todos los formularios y campos de entrada fueron realizados con *Perl* y el módulo *CGI* que maneja POO (Programación Orientada a Objetos), la creación de dichos objetos se explica en la sección 2.4.5 referente al módulo *CGI* de *Perl* en donde se explica como generar código *HTML* desde *Perl*.

2.4 MySQL

MySQL es un Sistema de Administración de Base de Datos Relacional (*RDBMS*: *Relational Database Management System*). Está liberado bajo *GNU GPL* para ser software libre, dependiendo para lo que se le utilice. Funciona como una base de datos tipo servidor/cliente, lo cual quiere decir que se dispone de un servidor *Web daemon* que se ejecuta en segundo plano, a la escucha de las peticiones del programa cliente. En MySQL, el servidor *Web daemon* es *MySQLd* y el cliente *MySQL*. Actualmente es el servidor de base de datos más popular para los desarrollos *Web* debido a que es muy rápido y sólido.

Las operaciones y sintaxis de lenguaje que se mostrarán a continuación son usadas por el módulo *DBI* de *Perl*, el cual se documenta en la sección 2.4.13.

2.4.1 Uso de tablas de MySQL

Las tres funciones mostradas en la tabla 2.9, permiten crear o eliminar tablas, ó bien, modificar la estructura de las tablas agregando o eliminando campos.

Tabla 2.9 Funciones para la creación de tablas.

CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Creación de tablas

La función utilizada para crear una nueva tabla debe utilizar una sentencia cuya sintaxis sea como la siguiente:

CREATE TABLE **Tabla** (**Campo Tipo** (**Tamaño**),…);

En donde:

Tabla: Es el nombre de la tabla que se va a crear.

Campo: Es el nombre del campo o de los campos que se van a crear en la nueva tabla, debe contener al menos uno.

Tipo: Es el tipo de datos de campo en la nueva tabla.

Tamaño: Es el tamaño del campo que sólo se aplica para campos de tipo texto.

Como ejemplo se muestra el siguiente enunciado, que crea una nueva tabla llamada **Persona** con dos campos, uno llamado **ID** de tipo Entero (*integer*) y longitud de 5 caracteres y otro llamado **Nombre** de tipo cadena (*varchar*) con longitud 25 caracteres:

CREATE TABLE **Persona** (**ID** *INTEGER* (5), **Nombre** *VARCHAR* (25));

Eliminación de tablas

La función utilizada para eliminar una tabla debe utilizar una sentencia cuya sintaxis sea como la siguiente:

DROP TABLE **tabla**;

Como ejemplo se muestra el siguiente enunciado, el cual elimina la tabla llamada **Persona**:

DROP TABLE **Persona**;

Modificación de tablas

La función utilizada para Modificar el diseño de una tabla, agregando o eliminando una columna, utiliza una sentencia como la siguiente:

ALTER TABLE **Tabla** {*ADD* {*COLUMN* **campo tipo (tamaño)** | *DROP* {*COLUMN* **campo**}};

En donde:

Tabla: Es el nombre de la tabla que se desea modificar.

Campo: Es el nombre del campo que se va a añadir o eliminar.

Tipo: Es el tipo de campo que se va a añadir.

Tamaño: Es el tamaño del campo que se va a añadir (sólo para campos de texto).

Las operaciones que se pueden realizar se muestran en la tabla 2.10. Como ejemplo se muestra el siguiente enunciado, que agrega a la tabla **Persona** un campo denominado **Apellido** de tipo cadena (*varchar*), con una longitud de 23 caracteres:

ALTER TABLE **Persona** *ADD COLUMN* **Apellido** *VARCHAR* (25);

Y el siguiente enunciado que elimina el campo denominado **Apellido**

ALTER TABLE **Persona** *DROP COLUMN* **Apellido**;

Tabla 2.10 Funciones básicas para la modificación de la estructura de tablas.

ADD COLUMN	Se utiliza para añadir un nuevo campo a la tabla, indicando nombre, tipo de campo y opcionalmente tamaño para campos de tipo texto.
ADD	Se utiliza para agregar un índice de multicampos o de un único campo.
DROP COLUMN	Se utiliza para borrar un campo cuyo nombre se específica.
DROP	Se utiliza para eliminar un índice. Se especifica únicamente el nombre del índice a continuación de la palabra reservada *CONSTRAINT*.

2.4.2 Consultas a una base de datos

Normalmente se realizan cuatro principales tipos de consulta cuando se tiene acceso a una base de datos: seleccionar, insertar, actualizar y eliminar un renglón de datos; ó bien solo uno, ó un grupo de datos que cumplan ciertas condiciones. Estas funciones se muestran en la tabla 2.11.

Tabla 2.11 Funciones básicas para consultas

SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados.
DELETE	Utilizado para eliminar registros de una tabla de una base de datos.

Selección de registros

La función *SELECT* se utiliza para realizar consultas de selección e indica al motor de datos que devuelva información de las bases de datos. Esta información es devuelta en forma de conjunto de registros que se pueden almacenar. La sintaxis básica de una consulta de selección es la siguiente:

SELECT **Campos** *FROM* **Tabla**;

En donde **Campos** es la lista de campos que se deseen recuperar y **Tabla** es el origen de los mismos, por ejemplo el siguiente enunciado de consulta devuelve una cadena con el campo **Nombre** y **Teléfono** de la tabla **Clientes**:

SELECT **Nombre**, **Teléfono** *FROM* **Clientes**;

Actualización de registros

La función *UPDATE* se utiliza para crear una consulta de actualización que cambia los valores de los campos de una tabla basándose en un criterio específico. Su sintaxis es:

UPDATE **Tabla** *SET* **Campo1**=**Valor1** *WHERE* **Criterio**;

UPDATE no genera ningún resultado. Para saber qué registros se van a cambiar, hay que examinar primero el resultado de una consulta de selección que utilice el mismo criterio y después ejecutar la consulta de actualización. Por ejemplo el siguiente enunciado que actualiza todos los registros de la columna **Grado** que tengan el valor de 2 con un valor de 5.

UPDATE **Empleados** *SET* **Grado** = 5 *WHERE* **Grado** = 2;

Eliminación de registros

La función *DELETE* se usa para crear una consulta de eliminación que elimina los registros de una o más de las tablas listadas en la cláusula *FROM* que satisfagan la cláusula *WHERE*. Una consulta de borrado elimina los registros completos, no únicamente los datos en campos específicos. Si se desea eliminar valores en un campo especificado, se debe crear una consulta de actualización que cambie los valores a *NULL*. Una vez que se han eliminado los registros utilizando una consulta de borrado, no se puede deshacer la operación. Si se desea saber qué registros se eliminarán, primero se debe examinar los resultados de una consulta de selección que utiliza el mismo criterio y después ejecutar la consulta de borrado.

DELETE * *FROM* **Empleados** *WHERE* **Cargo** = **'Vendedor'**;

Agregar registros

La función *INSERT INTO* agrega un registro en una tabla. Se le conoce como una consulta de datos añadidos. Para insertar un único registro en este caso la sintaxis es la siguiente:

INSERT INTO **Tabla** (**campo1**) *VALUES* (**valor1**);

Esta consulta actualiza el campo1 con el valor1. Hay que prestar especial atención a acotar entre comillas simples (') los valores literales (cadenas de caracteres) y las fechas

indicarlas en formato mm-dd-aa y entre el carácter (#). Por ejemplo el siguiente enunciado que agrega un registro en la columna **Grado** que tengan el valor de **5**.

INSERT INTO Empleados (**Grado**) *VALUES* (**5**);

2.5 Perl

Perl (*Practical Extraction and Report Language*) se deriva del lenguaje C de programación. Es un lenguaje interpretado, por lo tanto no compilado, así que los programas se mandan a un intérprete que divide la ejecución en dos fases: tiempo de compilación y tiempo de ejecución. En tiempo de compilación el interprete parsea el texto del programa en un árbol sintáctico. En tiempo de ejecución, ejecuta el programa siguiendo el árbol [Perl 2007].

Las facilidades para la manipulación de procesos, archivos y texto hace que este lenguaje sea usado en tareas donde se involucran el rápido desarrollo de programas, desarrollo de utilerías para el sistema operativo, herramientas de software, tareas relacionadas con la administración de sistemas, manejo de bases de datos, programación de gráficas, de redes, y de Internet.

2.5.1 Estructura del lenguaje Perl

Al inicio de un programa de *Perl* se debe indicar donde se encuentren instalado el archivo binario de *Perl* utilizando la estructura: **#!*ubicación***, en *Linux* usualmente este enunciado es:

#!*/usr/local/bin/perl*

Los nombres de los archivos de programas en *Perl*, por convención, finalizan con la terminación **.pl** en *Unix* y la extensión **.pl** en *MS-DOS*. Cada línea de sentencia, instrucción debe separarse de otra con punto y coma (**;**). Las líneas de comentarios deben iniciar con el símbolo (**#**). Los bloques de código de *Perl*, tales como los ciclos de control y las condiciones siempre deben encerrarse entre llaves (**{}**).

2.5.2 Escalares y arreglos de escalares

El dato de tipo escalar es el dato básico de *Perl* y puede ser un entero, punto flotante, o una cadena. Las variables escalares siempre tiene que iniciar con el símbolo prefijo **$**. Las variables en *Perl* no tienen que ser declaradas al inicio del programa, como ocurre con el lenguaje C. Además dichas variables se teclean y se evalúan en base al contexto del programa. Por ejemplo:

```
$x=4;      # un entero
$y="11";  # una cadena
$z= 4.5;  #de punto flotante
```

Para el uso de arreglos siempre tienen que iniciar con el símbolo prefijo **@**. El primero de los elementos de un arreglo siempre ocupa la localidad cero, pueden contener datos de diferente tipo y son seleccionados individualmente mediante el uso de índices. Los arreglos son una lista, que es una secuencia de valores escalares dentro de un paréntesis. Por ejemplo:

```
@arreglo = (4,5, "Perl", 2);
```

Las funciones de *Perl* para manejar arreglos son:

- *pop*: Extraer el escalar del final del arreglo.
- *push*: Insertar un escalar al final del arreglo.
- *shift*: Extraer el escalar del inicio del arreglo.
- *unshift*: Insertar un escalar al inicio del arreglo.

2.5.3 Estructuras de selectividad y repetividad

Las estructuras selectivas son aquellas que evalúan una condición y en función del resultado de la misma se ejecuta una instrucción u otra. Una de las estructuras selectivas y la más usada son la definida como ***if*** y sus cláusulas consecutivas ***elsif*** y ***else***. Que en la sintaxis del lenguaje *Perl*, es:

```
if( condición ) {
  ...
}elsif( otra condición ) {
  ...
}else {
  ...
}
```

Como estructuras de iteración de una secuencia de instrucciones *Perl* tiene varias implementadas, como son: ***while***, ***unless***, ***for, do*** y ***foreach***. Se presenta solamente la sintaxis en *Perl* para *while* y *for* en la tabla 2.12.

Tabla 2.12 Estructura de control para *while* y *for*

while (condición) {
…
}
	for ($i=0; $i <= $max; $i++) {
	…
	}

2.5.4 Funciones

Perl también incluye distintas funciones, algunos ejemplos son:

- *chomp*: divide una cadena usando un delimitador de la expresión.
- *chop*: elimina el último carácter de una cadena.
- *lc*: convierte en minúsculas una cadena.
- *length*: regresa el número de *bytes* de una cadena.
- *local*: crea un valor temporal para una variable global.
- *my*: declara y asigna una variable local.
- *printf*: Imprime en la salida una lista formateada.
- *print*: Imprime en la salida una lista.
- *split*: secciona una cadena utilizando un carácter como delimitador.

2.5.5 Módulo CGI de Perl

El módulo *CGI* (*Common Gateway Interface*) es una biblioteca que usa *Perl* para facilitar la creación de formularios *Web* y analizar su contenido gramaticalmente. El módulo provee funciones abreviadas que producen código *HTML*, reduciendo los errores de codificación. También provee funcionalidad para algunas características más avanzadas como son:

soporte para carga de archivo, *cookies*, *CSS* (*cascade style sheets*) y marcos. Existen dos estilos de programación usando el módulo *CGI*: orientado a objetos y orientado a funciones, solamente se presenta el estilo orientado a objetos [PerldocCGI 2006].

El estilo orientado a objetos crea uno o más objetos *CGI* y entonces se usan métodos objeto para crear los elementos de la página. Cada objeto se inicia con la lista de los nombres de parámetros que fueron pasados al módulo por el servidor. Se pueden modificar los objetos, guardarlos en un archivo o base de datos y recrearlos. Para crear una nueva instancia de objeto se realiza como sigue:

$q=*new CGI*;

Esto analizará las entradas (para ambos métodos *POST* y *GET*) y la guardará como una instancia de objeto de Perl llamado $*q*.

2.5.6 Generación de documentos dinámicos

La mayoría de las funciones del módulo permiten la creación dinámica de documentos. Cada función produce un fragmento de HTML ó *HTTP* que se puede mostrar directamente.

Para crear el encabezado *HTTP*, que es la primera tarea que se realiza en cualquier *script CGI*, se utiliza la función *header*(). Esta función regresa el *Content-type*: *header*, y se declara de la manera siguiente:

print $q->*header*();

Para crear el encabezado de un documento *HTML* se utiliza la función *start_html*() mientras que la función *end_html* () regresa la etiqueta de cierre del documento </*html*>. Los cuales se declaran de la manera siguiente:

print $q->*start_html*();
...contenido del documento...
print $q->*end_html*();

Para iniciar un formulario se utiliza la función *start_form*() la cual codifica los argumentos y le dice al navegador como empaquetar los campos antes de enviarlos al servidor. Se puede asignar el método de envío de los datos usando el parámetro *method*. La función *end_form*() regresa la etiqueta de cierre del formulario *</form>*. Las funciones se declaran de la manera siguiente:

```
print $q->start_form();
        ...contenido del formulario...
print $q->end_form();
```

2.5.7 Creación de campos, menúes y botones

A continuación se presentan las funciones para crear elementos de formularios usando el módulo *CGI*, ó bien, usando código *HTML*. Los elementos que se presentan son: campos de textos, campos de texto múltiple, campos de contraseña, campos ocultos, casillas de verificación, menúes desplegables, listas desplegables, grupos de casillas de verificación y grupos de botones de opción.

Campos de texto

Para crear un campo de entrada de texto, se utiliza la función *textfield*(). La sintaxis es:

```
print $q->textfield( -name=>'usuario',
                     -value=>'root',
                     -size=>50,
                     -maxlength=>80 );
```

La alternativa usando sentencias en código *HTML*:

```
print " <input name=\"usuario\" type=\"text\" value=\"root\" size=\"50\"
maxlength=\"80\" > ";
```

Campos de texto múltiple

Para crear un campo de texto múltiple se utiliza la función *textarea*(), cuya sintaxis es:

```
print $q->textarea( -name=>'domicilio',
                    -default=>'Ocampo 719',
                    -rows=>10,
                    -columns=>50 );
```

Si se utilizara código *HTML* la sentencia es:

```
print " <textarea name=\"domicilio\" default=\"Ocampo 719\" rows=\"10\" cols=\"50\">";
```

Campos de contraseña

Para crear un campo de contraseña cuyo contenido será mostrado fuera de la página *Web*, se utiliza la función *password_field*(). Se utiliza la sintaxis siguiente:

```
print $q->password_field( -name=>'pwd',
                          -value=>'guest',
                          -size=>50,
                          -maxlength=>80);
```

Usando sentencias en código *HTML* y no la función:

```
print " <input name=\"pwd\" type=\"password\" value=\"guest\" size=\"50\"
maxlength=\"80\" > ";
```

Campos ocultos

Para crear un campo oculto que no puede ser visto por el usuario, se utiliza la función *hidden*(). Esto es útil para pasar información del estado de las variables de un *script* al siguiente, la sintaxis es:

```
print $q->hidden( -name=>'ID',
                  -default=>['1'] );
```

La alternativa usando sentencias en código *HTML*:

```
print " <input name=\"ID\"  type=\"hiddden\"  default=\"1\" > ";
```

Casillas de verificación

Para crear una casilla de verificación que no esta relacionadas con ninguna otra, se utiliza la función *checkbox*(). La sintaxis es:

```
print $q->checkbox(  -name=>'actualizar',
                     -checked=>1,
                     -value=>'actualizar',
                     -label=>'Desea actualizar' );
```

Usando sentencias en código *HTML* y no la función:

```
print " <input  name=\"actualizar\"  type=\"checkbox\"  value=\"actualizar\"
checked=\"1\"> Desea actualizar  ";
```

Menúes desplegables

Para crear un menú desplegable, se utiliza la función *popup_menu*(). La sintaxis es:

```
print $q->popup_menu(  -name=>'genero',
                       -values=>['Hombre','Mujer'],
                       -default=>");
```

La cual remplaza el uso de la sentencia en código *HTML*:

```
print " <select name = \"genero\">
<option value = \" \">  </option>
<option value = \"hombre\"> Hombre </option>
<option value = \"mujer\"> Mujer </option>
```

```
</select> ";
```

Listas desplegables

Para crear una lista desplegable, se utiliza la función *scrolling_list*(). La sintaxis es:

```
print $q->scrolling_list( -name=>'lista_usuarios',
                          -values=>['root','guest','ftzintzun'],
                          -default=>['root',' ftzintzun '],
                          -size=>3,
                          -multiple=>'true' );
```

Si se utilizara código *HTML* la sentencia es:

```
print " <select name = \"lista_usuarios\" size = \"3\" multiple>
<option value = \"root\" > root </option>
<option value = \"guest\"> guest </option>
<option value = \"ftzintzun\"> ftzintzun </option>
</select> ";
```

Casillas de verificación

Para crear un grupo de casillas de verificación relacionadas por el mismo nombre, se utiliza la función *checkbox_group*(). La sintaxis para declararlo es:

```
print $q->checkbox_group( -name=>' ID ',
                          -values=>[ '1' , '2' , '3' , '4' ],
                          -default=>[ '1', '4' ],
                          -linebreak=>'true' );
```

La alternativa usando sentencias en código *HTML*:

```
print " <input name=\"ID\"  type=\"checkbox\"   value=\"1\" checked"> 1 <p>";
print " <input name=\"ID\"  type=\"checkbox\"   value=\"1\" > 2 <p>";
```

```
print " <input name=\"ID\"  type=\"checkbox\"  value=\"1\" > 3 <p>";
print " <input name=\"ID\"  type=\"checkbox\"  value=\"1\" checked> 4 <p>";
```

Botones de opción lógicos

Para crear un grupo de botones de opción, se utiliza la función *radio_group*(). Esto crea un paquete de botones de opción lógicos relacionados (si se selecciona uno de elemento del grupo los otros miembros se deseleccionan). La sintaxis para declararlo es:

```
print $q-> radio_group(  -name=>'genero',
                         -values=>['Masculino', 'Femenino'],
                         -default=>'Femenino',
                         -linebreak=>'true' );
```

Y la alternativa usando sentencias en código *HTML*:

```
print " <input name=\"genero\"  type=\"radio\"  value=\"M \" checked> Masculino <p>";
print " <input name=\"genero\"  type=\"radio\"  value=\"F\"> Femenino <p>";
```

Botones de envío

Para crear un botón de petición de envío, se utiliza la función *submit*(). La sintaxis para declararlo es:

```
print $q->submit(  -name=>'action',
                   -value=>'Enviar');
```

Usando sentencias en código *HTML*:

```
print " <input type=\"submit\"  value=\"Enviar \">";
```

Botones de reinicio

Para crear un botón que restaura el formulario a sus valores anteriores, se utiliza la función *reset*(). La sintaxis para declararlo es:

```
print $q->reset;
```

O bien usando sentencias en código *HTML*:

```
print " <input type=\"reset\"  value=\"Borrar \">";
```

Botones default

La función *defaults*() crea un botón que al ser invocado ocasiona que los valores del formulario sean reiniciados a los valores por defecto, borrando los cambios hechos por el usuario. Esta función no tiene una sentencia equivalente en código *HTML*. La sintaxis que se utiliza es:

```
print $q->defaults('valores iniciales');
```

2.5.8 Utilización de Cookies y archivos CSS

Cookies

Las *cookies* sirven para guardar pequeñas partes de datos en el navegador que pueden ser pedidos y regresados al *script*. El módulo *CGI* las define como un par nombre=valor. Para crear una *cookie* se usa la función *cookie*() y la declaración de las cookies debe ser incorporado en el encabezado *HTTP*. Dentro de la cadena regresada por el método *header*(). De modo que el uso de los métodos sea como se muestra en el ejemplo siguiente:

```
$cookie = $q->cookie(  -name=>'usuario',
                       -value=>'guest',
                       -expires=>'+1h',
                       -path=>'/index.htm' ) ;
print $q->header(  -cookie=>$cookie ) ;
```

Para llamar la información de una *cookie* se usa su nombre como argumento mediante el método *cookie*(), sin usar ningún parámetro. Las *Cookies* regresan el tipo de valor original, aquellas que contengan un escalar, arreglo o *hash* regresaran esos mismos tipos de valores. Como se muestra en el ejemplo:

```
use CGI;
$q = new CGI;
$usuario = $q->cookie('usuario');
```

Carga de archivos CSS

El módulo *CGI* de *Perl* tiene un soporte limitado para hojas de estilo (*CSS*). Para incorporarse dentro de los documentos, se usa el método *start_html*() mediante el parámetro *style*. La forma más recomendable de usar estilos es creando un archivo *CSS* y agregarlo de la manera siguiente:

```
print $q->start_html( -base=>'true',
                      -target=>'_self',
                      -style=>{ 'src'=>styles.css' } );
```

2.5.9 Recepción de parámetros en el script

Para capturar el valor de los parámetros recibidos se puede realizar de dos maneras usando el módulo *CGI* ó consultando las variables de entorno. Usando el módulo *CGI* se utiliza la función *param*(), la cual captura los datos conservando el tipo de datos solamente usando el nombre del campo de la manera siguiente:

```
$variable = $q->param ('nombre del campo');
```

La otra forma consiste en consultar la variable de entorno *REQUEST_METHOD*. Su contenido será *POST* ó *GET* según sea el caso:

```
If($ENV{'REQUEST_METHOD'} eq 'GET') {…}
If($ENV{'REQUEST_METHOD'} eq 'POST') {…}
```

Los datos enviados con *POST* llegaran al *script* a través de la entrada estándar y la longitud de la cadena de datos que arribarón estará almacenada en la variable de entorno *CONTENT_LENGTH*.

2.5.10 Módulo DBI de Perl

DBI (*DataBase Independent interface*) es un módulo de acceso a bases de datos para *Perl*. Define un paquete de métodos, variables y convenciones que proveen una interfaz de bases de datos consistente e independiente de la base de datos se este utilizando.

Tiene soporte para trabajar con bases de datos como Oracle, Sybase, Informix, MySql, mSql, bases de datos con soporte ODBC (MS-Access, SQL Server), etc. Actualmente *DBI* sólo trabaja con bases de datos relacionales y no con bases de datos orientadas a objetos [PerldocDBI 2006].

Conectarse y desconectarse de una Base de Datos

Para trabajar con DBI es necesario inicialmente cargar el módulo DBI:

use DBI;

Después y para cualquier tarea que se desee realizar hay que abrir una conexión con la base de datos. La información que se utiliza para establecer la conexión es:

- **Nombre de la fuente de datos:** Esto es una cadena que contiene los siguientes elementos, separados por dos puntos:
  - *DBI*
  - Nombre del *driver* (manejador). Se puede usar diferentes drivers, por ejemplo: ODBC, MySQL, ORACLE, etc.
  - Nombre de la base de datos con la que se esta trabajando.

  Por ejemplo:

  $*dsn* = "*DBI*:*MySQL*:constructora:*localhost*";

- **Usuario**: Nombre del usuario que se va a conectar a la base de datos.
- ***Password***: Clave de seguridad del usuario.

- *Host* (opcional): Es la máquina donde se encuentra la base de datos.
- **Puerto** (opcional): Es el puerto que se utiliza para conectarse a la base de datos.

Por ejemplo:

```
$dbh = DBI->connect($dsn, $usuario, $password) or die $DBI::errstr;
```

Una vez que se ha terminado de trabajar con la base de datos se debe cerrar la conexión con la base de datos, no es estrictamente necesario, pero si aconsejable, ya que algunas bases de datos pueden no terminar sus transacciones correctamente. Se utiliza el método *disconnect* como se muestra:

```
$dbh->disconnect;
```

En caso de un error en la conexión el valor de retorno es *undef*. Para capturar el error se utiliza el método $DBI::err ó $DBI::errstr.

Funciones del módulo DBI

Para realizar operaciones con bases se usan declaraciones que usan la estructura del lenguaje SQL, usado por MySQL y que fue mostrado en la sección 2.3.1 y 2.3.2, mediante los métodos *prepare* y *do*.

El método *prepare*() permite hacer una declaración para su posterior uso. Como ejemplo se realiza la consulta *SELECT*:

```
$sth = $dbh->prepare(" SELECT * FROM tabla WHERE id = ? ");
$sth->execute( $id );
while ( @renglon = $sth->fetchrow_array ) {
        print "@ renglon \n";
}
```

El método *do*() puede ser usado para las declaraciones que son realizada una sola vez. Como ejemplo se realiza la actualización de un registro mediante *UPDATE*:

```
$rows_affected = $dbh->do("UPDATE tabla SET registro = registro + 1");
```

Capítulo 3:

Modelo del catálogo de datos

"*No nos preguntemos si podemos conocer la naturaleza del universo, la vía láctea, una estrella o un mundo, sino si nos es dado conocer, en ultima instancia y de forma pormenorizada, la naturaleza de un grano de sal*"
Carl sagan - El cerebro de broca

"Entonces dijo Dios: Hagamos al hombre a nuestra imagen, conforme a nuestra semejanza"
Génesis 1:27

Capítulo 3:

Modelo del catálogo de datos

En el presente capítulo se presentan los conceptos básicos necesarios para implementar un modelo de sistema de bases de datos orientadas a objetos (BDOO). Estos conceptos consisten de forma breve, en los siguientes puntos:

1. Conceptos básicos del sistema orientando a objetos
2. Enfoques para la construcción de sistemas BDOO
3. Lenguajes de gestión de datos, que son usados para la administración y manipulación de objetos y clases en el sistema.
4. Las relaciones de clases propuestas para la implementación del sistema.

3.1 Conceptos básicos del sistema orientado a objetos

Las bases de datos orientadas a objetos (BDOO) almacenan y manipulan información que puede ser representada por objetos; además proporcionan una estructura flexible con acceso ágil, rápido y con gran capacidad de modificación. Esto se debe a que las funciones se descentralizan en todos los objetos que las componen y a su vez cada objeto realiza una función específica sin importar las demás funciones realizadas por otros objetos. Son más seguras ya que no permiten tener acceso a los objetos, sino solo a través de los métodos predeterminados. Además combina las mejores cualidades de los archivos planos, de las bases jerárquicas y de las bases de datos relacionales. Además las BDOO ofrecen un mejor rendimiento de la máquina que las bases de datos por relación, para aplicaciones ó clases con estructuras de datos complejas.

Sin embargo, las BDOO coexistirán con las bases de datos relacionales como una forma de estructura de datos dentro de una BDOO [INFORMATICO 2006].

3.1.1 Conceptos fundamentales

Para una mejor comprensión de los conceptos básicos de una base de datos orientada a objetos, se presentan separados en tres grupos fundamentales:

1. Objetos
2. Clases
3. Clases genéricas.

Objetos

- **Objeto:** Es cualquier cosa concreta sobre la que se almacena información y métodos que los controlan. Por ejemplo: en una empresa CONSTRUCTOR es aplicado a los objetos que son personas empleadas por una organización, alguna INSTANCIA podría ser Juan Pérez, María Sánchez etc.
- **Tipo de Objeto:** Es una categoría de objeto. Por ejemplo: CONSTRUCTOR.Un objeto es una Instancia de un tipo de objeto. Por ejemplo PERSONA (Juan Pérez).
- **Encapsulado:** Es la ocultación de los detalles de implementación de un objeto respecto al usuario.
- **Solicitud**: Es la petición de una operación específica, con uno ó más objetos como parámetros. Es decir, es para que se lleve acabo la operación indicada, llamada método, y se produzca un resultado. En consecuencia las implantaciones se refieren a los objetos como solicitudes.
- **Método:** es una lista de instrucciones detalladas que definen cómo responde un objeto a un mensaje en particular.

Clases

- **Clase:** Es una implantación de un tipo de objeto. Especifica una estructura de datos y los métodos operativos permisibles que se aplican a cada uno de sus objetos, siendo un conjunto de objetos que tienen un mismo comportamiento (comparten una misma funcionalidad) que se puede observar desde afuera.
- **Herencia:** Una clase implanta el tipo de objeto. Una Subclase hereda propiedades de su clase padre, la estructura y todos sus métodos ó algunos de ellos.

Clases genéricas

- **Genericidad:** Permite construir clases genéricas para otras clases.
- **Objetos Complejos:** Están construidos mediante objetos más simples ó mediante la aplicación de constructores. Los Objetos más simples son por ejemplo: enteros, caracteres, cadenas, booleanos ó punto flotante.

3.1.2 Notación

En las referencias existen diversas notaciones referentes a bases de datos y bases de datos orientados a objetos, las cuales resultan ser muy similares en algunos conceptos; pero sumamente contrastantes con la notación informal que se les suele asignar a los objetos, por lo cual, se propone la notación mostrada en la tabla 3.1 y que a partir de este capítulo será utilizada.

Tabla 3.1 Notación formal e informal referentes a bases de datos.

Notación Informal	Notación Formal	
	Bases de Datos	Bases de Datos Orientadas a Objetos
Tabla	Relación	Clase
Campo	Atributo	Atributo
Tablas Relacionadas	*Join*	Relaciones: • Atribución (Tiene) • Inclusión (es un) • Tupla • Conjunto
Registros (renglón)	Tuplas	Objetos

3.2 Enfoques para la construcción de sistemas BDOO

Para la construcción de BDOO, pueden ser utilizados algunos de los dos enfoques siguientes [INFORMATICO 2006]:

1. Se puede utilizar el código actual altamente complejo de los sistemas de administración de las bases de datos relacionales, de modo que una BDOO se implemente más rápido sin tener que iniciar de cero. Las técnicas orientadas a

objetos se pueden utilizar como medios para el diseño sencillo de sistemas complejos. Los sistemas se construyen a partir de componentes ya probados con un formato definido para las solicitudes de las operaciones del componente.

2. Se considera a una BDOO como una extensión de la tecnología de las bases de datos relacionales. De este modo, las herramientas, técnicas, y vasta experiencia de la tecnología de las bases de datos relacionales se utilizan para construir un nuevo sistema administrador de base de datos. Se pueden añadir apuntadores a las tablas de relación para ligarlas con objetos binarios de gran tamaño. La base de datos también debe proporcionar a las aplicaciones clientes un acceso aleatorio y por partes a grandes objetos, con el fin de que sólo sea necesario recuperar a través de la red la parte solicitada de los datos.

El segundo enfoque de construcción de BDOO es el que fue el implementado para la construcción del sistema, debido a que se utilizaron las herramientas de bases de datos relacionales proporcionadas por *MySQL* y el lenguaje de programación *Perl*.

3.3 Lenguajes de gestión de datos

La propuesta del sistema (SGUDEH) consiste en implementar un sistema que sea lo suficientemente flexible al manejar bases de datos. Para lograr este objetivo es necesario proveer alguna forma de los tres lenguajes básicos utilizados en los sistemas de gestión de datos SQL [Networksolutions 2007]: lenguaje de definición de datos (*DDL*), lenguaje de manipulación de datos (*DML*) y lenguaje de control de datos (*DCL*).

El propósito del lenguaje de definición de datos (*Data Definition Lenguaje*, *DDL*), en el SGUDEH, es el de facilitar la portabilidad de los esquemas de las bases de datos. Como el lenguaje *DDL* no es un lenguaje de programación completo, solo debe definir las propiedades y los prototipos de las operaciones de los tipos, pero no los métodos que implementan esas operaciones [DBAsupport 2007]. Por lo cual, debe realizar con cada clase y sus relaciones las siguientes operaciones:

1. Definir clases
2. Modificar atributos de cada clase
3. Eliminar atributos y clases

El lenguaje de manipulación de datos (*Data Manipulation Lenguaje*, *DML*) en el SGUDEH debe implementar funciones que realicen con cada clase y sus relaciones, las siguientes operaciones de consulta:

1. Inserción de objetos
2. Actualización de objetos
3. Borrado de objetos
4. Consulta de objetos.

El lenguaje de control de datos (*Data Control Lenguaje*, DCL) en el SGUDEH se emplea para gestionar el acceso a los datos para un entorno multiusuario, en la que se presenta la protección de datos, seguridad de clases, establecimiento y restricciones de acceso, proponiéndose las siguientes operaciones:

1. Creación de usuarios
2. Creación de bases de datos
3. Seguridad del sistema.

3.4 Relaciones de clases para el uso del sistema semántico

Las relaciones para el sistema semántico son implementadas por el lenguaje DDL del sistema, para lo cual, se deben señalar de forma especifica los métodos para establecer las relaciones de clases. Estos métodos están basados en la propuesta original [Antunes 1995] y los cuales son:

1. Clases genéricas.
2. Atribución.
3. Inclusión.
4. Composición.

3.4.1 Clases genéricas

Una clase está compuesta por distintos atributos y sus componentes deben estar definidos por objetos genéricos, para lo cual, se implementa la creación de una clase fundamental conocida como clase genérica.

El propósito de las clases genéricas es el de definir los atributos de una clase modelándola detalladamente con elementos conocidos. Las clases genéricas están compuestas por objetos genéricos, los cuales son llamados comúnmente valores primitivos [Antunes 1995].

Para el sistema, estas clases son implementadas mediante un catálogo de clases genéricas. El cual permite implementar básicamente seis valores primitivos, como se observa en la figura 3.1:

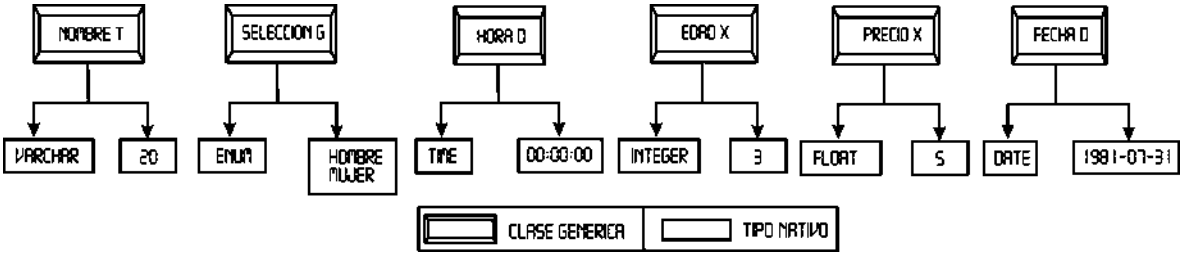


Figura 3.1 Representación de clases genéricas.

1. ***Integer:*** Números enteros.
2. ***Time:*** Hora. La cual tiene el formato: Horas:Minutos:Segundos, implementando como valor por defecto: 00:00:00.
3. ***Varchar:*** Cadenas de caracteres. Se utilizado para crear secciones de texto de una longitud de n-caracteres.
4. ***Enum:*** Utilizado para seleccionar entre diferentes valores.
5. ***Float:*** Números de punto flotante.
6. ***Date:*** Fecha. El cual tiene el formato: Año-Mes-Día, implementando como valor por defecto: 0000-00-00.

3.4.2 Atribución

Esta relación permite la creación de una clase, así como la asignación de propiedades descriptivas mejor conocidos como atributos [Antunes 1995]. Los atributos pueden ser creados de dos formas:

1. Atributos creados a partir de una clase genérica
2. Atributos creados a partir de una clase.

Del ejemplo base: una persona tiene nombre, un apellido paterno así como también tiene un grupo de obras compradas como atributo, como se observa en la figura 3.2.

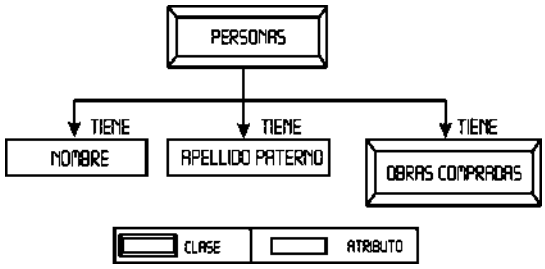


Figura 3.2 Representación de atribución.

3.4.3 Inclusión

Esta relación es utilizada para la asignación de relación entre clases, lo cual da surgimiento a la noción de superclase o clase padre y subclase o clase hija de una clase dada [Antunes 1995]. En este caso de relación se puede presentar la herencia múltiple de clases, es decir, que una subclase tiene más de una clase padre. Permitiendo así que una subclase herede los atributos de dos o más clases diferentes.

Del problema base propuesta en la sección 1.4, considere: un constructor es una persona en donde constructor es una subclase de persona y a la vez persona es una superclase de constructor, como se observa en la figura 3.3.

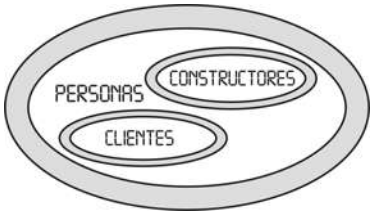


Figura 3.3 Representación de inclusión.

3.4.4 Composición

Esta relación permite la denotación de una relación en clases específicas, dando surgimiento a la noción de dos clases de volumen:

1. Tupla
2. Conjunto

Las instancias de estas dos clases son objetos construidos por los métodos agregación y relación.

Tupla

La clase específica tupla consiste en la relación de objetos distintos de un universo de objetos de dos clases diferentes dadas [Antunes 1995]. Por ejemplo, un expediente esta compuesto por la asignación de la documentación de una obra aun constructor, como se observa en la figura 3.4.

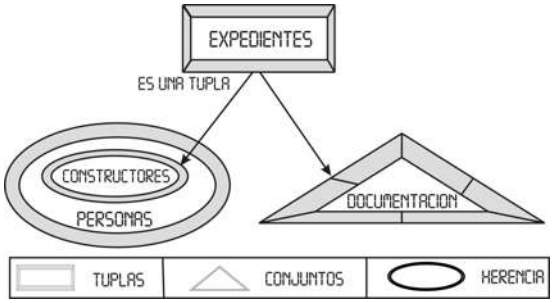


Figura 3.4 Representación de tupla.

Conjunto

La clase específica conjunto consiste en la extracción sin ordenar de un conjunto de objetos de un universo de objetos de una clase y su agrupación en una nueva clase [Antunes 1995]. Por ejemplo, Fraccionamientos es una instancia de grupo de casas y es construido como una colección de casas sin ordenar. Del mismo modo, los centros comerciales son una instancia de grupos de locales y son construidos como una colección de locales sin ordenar, como se observa en la figura 3.5.

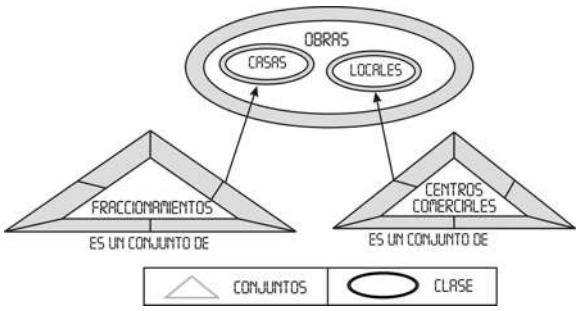


Figura 3.5 Representación de un conjunto.

Capítulo 4

Implementación del Sistema

"*Vé, pues, ahora, y escribe esta visión en una tabla delante de ellos, y regístrala en un libro*"
Isaías 30:8

"*There could have been no two hearts so open, no tastes so similar, no feelings so in unison*"
The lake house - Kate reading persuasion

Capítulo 4

Implementación del Sistema

El presente capítulo presenta el método propuesto para la implementación de los conceptos de relación de clases expuestos en el capítulo 3, asociando cada concepto de relación a su implementación mediante tablas relacionales.

4.1 Relaciones a implementar en el sistema semántico

Los métodos que integran el sistema (SGUDEH) implementan las clases necesarias para establecer las relaciones expuestas en la sección 3.4, y que son:

1. Clases genéricas
2. Atribución.
3. Inclusión.
4. Composición por tupla.
5. Composición por conjunto.

Para facilitar el uso de las sentencias de los operadores semánticos se propone un catálogo de datos, como el mostrado en la figura 4.1, donde se seleccionen las opciones de relación para crear las instancias de clases y se puedan seleccionar las clases genéricas para definir atributos.

Figura 4.1 Catálogo de datos.

4.1.1 Implementación de clases genéricas

Las clases genéricas son necesarias para asignar un atributo a una clase, la cual hereda a los atributos los tipos nativos de las bases de datos. El principal propósito de las clases genéricas es la reutilización del mismo código que contiene el tipo nativo *SQL*, en la escritura de las diferentes sentencias en el catálogo de datos. Para la implementación se propone un catálogo de clases genéricas, el cual puede ser construido a través de formularios, como se muestra en la figura 4.2, cada clase genérica tiene tres datos:

- Nombre de la clase.
- Tipo nativo.
- Valores necesarios para implementar el tipo nativo.

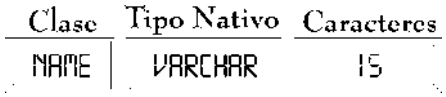


Figura 4.2 Implementación de clases genéricas.

Otras funciones que permite el formulario son actualizar y eliminar. La información de las clases genéricas debe ser almacenada en una tabla que cuente con tres campos, como se muestra en la figura 4.3.

CLASES GENERICAS

NOMBRE	TIPO	CARACTERES
NOMBRE T	VARCHAR	20
SELECCION G	ENUM	'HOMBRE', 'MUJER'
HORA D	TIME	
EDAD X	INTEGER	3
PRECIO X	FLOAT	5
FECHA D	DATE	

Figura 4.3 Almacenamiento de los datos de las clases genéricas.

De modo que las clases genéricas sean almacenadas con un nombre único, mientras que dependiendo del tipo nativo la información sea almacenada de la siguiente manera:

1. **Varchar**, se guarda junto con la cantidad máxima de caracteres para almacenar.
2. **Enum**, se guarda junto con la cadena de selección de valores. Cada valor debe ir entre comilla simple y separada por comas.

3. **Time**, no se incluye ningún valor. El valor por defecto (00:00:00) es implementado por el método.
4. **Integer**, se guarda junto con la cantidad máxima de caracteres por almacenar.
5. **Float**, se guarda junto con la cantidad máxima de caracteres por almacenar.
6. **Date**, no se incluye ningún valor. El valor por defecto (0000-00-00) es implementado por el método.

4.1.2 Implementación de atribución

Este caso consiste en asignar propiedades descriptivas a las clases, asociando esta función con el operador semántico **tiene**. Para la implementación mediante el catálogo de datos el cual es construido a través de formularios como se muestra en la figura 4.4, se solicitan los datos siguientes:

- Nombre de la clase existente o a crear.
- Operador semántico **tiene**.
- Nombre del atributo.
- Nombre de la clase genérica.

Del ejemplo base una persona tiene nombre, de manera que al utilizar el catálogo de datos la sintaxis de la sentencia debe ser como se muestra en la figura 4.4.

Figura 4.4 Implementación de atribución en el catálogo de datos.

Para implementar un atributo se verifica la existencia de la tabla; si esta no existe entonces se crea. A continuación en el ejemplo, al crearse la tabla **persona** se implementa un campo de apuntador usualmente conocido como *key*, al cual se le denomina ID. Se debe señalar que no se debe implementar un atributo con el nombre ID. El campo de apuntador (ID) es sumamente importante debido a que facilitará la implementación de los otros métodos, por lo tanto, deben contar con las siguientes características:

- Se incrementa en uno al insertarse un nuevo objeto a la clase.
- Debe ser único para cada objeto.

Continuando con el ejemplo, una vez creada la tabla **persona** se implementa el atributo **nombre**, siempre y cuando vayan acompañados de una clase genérica. De esta manera el método implementa una tabla como la mostrada en la figura 4.5.

PERSONA

ID	NOMBRE	APELLIDO	EDAD
1	JOSE	HERNANDEZ	25
2	EUNICE	DIAZ	28
3	GERARDO	CHORA	33
4	SAUL	CIPREZ	34

Figura 4.5 Implementación de atributos de clases.

El uso de este método evita la ejecución de muchas sentencias *SQL*, por ejemplo para generar una tabla como la mostrada en la figura 4.5, el código correspondiente debe ser:

CREATE TABLE persona (id *INTEGER* (15));

ALTER TABLE persona *ADD COLUMN* nombre *VARCHAR* (20);

ALTER TABLE persona *ADD COLUMN* apellido *VARCHAR* (20);

ALTER TABLE persona *ADD COLUMN* edad *INTEGER* (3);

Al usar la función **tiene** se debería que usar las sentencias

Persona **tiene** nombre nombret

Persona **tiene** apellido nombret

Persona **tiene** edad edadt

Con las clases genéricas correspondientes previamente definidas, en donde:

nombret es un *varchar* de 20 caracteres

edadt es un *integer* de 3 caracteres

Existe el caso de asignar una clase como atributo, entonces se crea un campo de *keys* con el nombre de la clase el cual permitirá guardar una ID de la clase, por ejemplo: **Obras tiene estilo**, como se muestra en la figura 4.6.

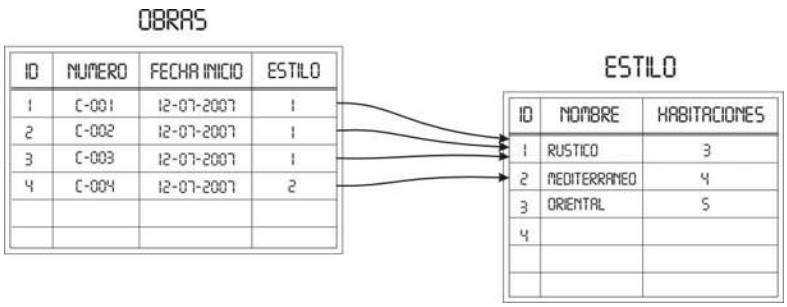


Figura 4.6 Implementación de clases como atributos.

La relación de atributo se guarda en un indice que contendrá todas las relaciones de clases como atributos, como se muestra utilizando el ejemplo anterior en la figura 4.7.

Figura 4.7 Implementación de clases como atributos en el índice.

4.1.3 Implementación de inclusión

La relación de herencia entre clases, crea una instancia de una subclase a partir de una clase. Se asocia esta función con el operador semántico **es**. Para la implementación mediante el catálogo de datos, como se muestra en la figura 4.8, se asignan en el formulario los datos siguientes:

- Nombre de la subclase.
- Operador semántico **es**.
- Nombre de la superclase.

Utilizando el ejemplo base: **constructor es persona**, entonces **constructor** representa una subclase de la clase **persona**, siendo la sintaxis de la sentencia como se muestra en la figura 4.8.

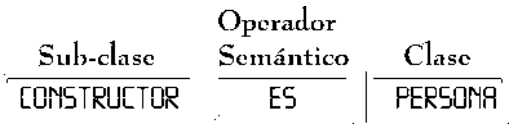


Figura 4.8 Implementación de especialización en el catálogo de datos.

Al implementar la herencia se verifica la existencia de la tabla **constructor**; si esta no existe entonces se crea. Entonces se implementa un campo de apuntador usualmente conocido como *key*, al cual se le denomina ID. El campo de apuntadores (ID) es sumamente importante debido a que facilita la implementación de los otros métodos, por lo tanto, deben contar con las siguientes características:

- Se incrementa en uno al insertarse un nuevo objeto a la clase.
- Debe ser único para cada objeto.

De la misma manera, antes de asignar herencia se debe verificar si existe la superclase para continuar. Al crear la tabla denominada **constructor**, a la cual se le pueden asignar más campos (atributos), se le relaciona con la tabla **persona** implementando una columna de apuntadores (*key*) con el nombre de la superclase. Esta columna se usa al escribir datos en la tabla **constructor** y se le asigna el valor del apuntador (ID) de la tabla **persona** donde se guarda la información correspondiente, como se muestra en la figura 4.9.

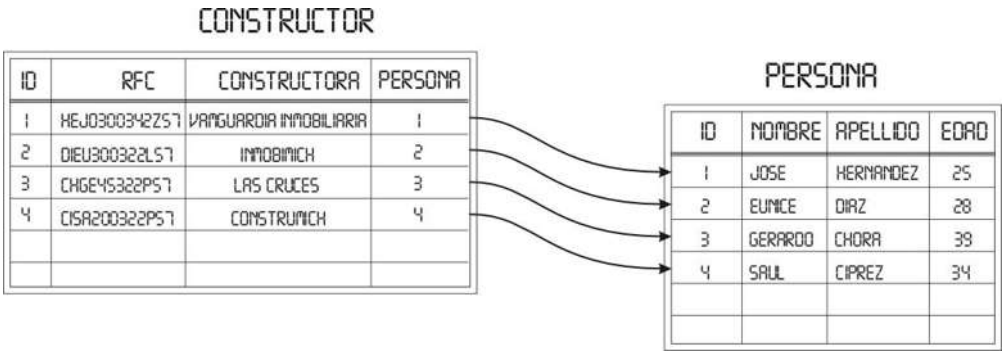


Figura 4.9 Implementación de clase mediante especialización.

Las relaciones deben registrarse en un índice (*index*) de relaciones, para facilitar la búsqueda de tablas relacionadas y que se pueda implementar fácilmente la herencia múltiple, donde, de acuerdo al tipo de relación se agrega un indicador numérico. Para este caso se le asigna el indicador 0, como se muestra en la figura 4.10.

Figura 4.10 Implementación de especialización en el índice.

Este método evita la ejecución de muchas sentencias *SQL*, por ejemplo para crear una tabla como la mostrada en la figura 4.9, el código correspondiente debe ser:

CREATE TABLE constructor (id *INTEGER* (15));

ALTER TABLE constructor *ADD COLUMN* rfc *VARCHAR* (9);

ALTER TABLE constructor *ADD COLUMN* horario *VARCHAR* (20);

ALTER TABLE constructor *ADD COLUMN* nombre *VARCHAR* (20);

ALTER TABLE constructor *ADD COLUMN* apellido *VARCHAR* (20);

ALTER TABLE constructor *ADD COLUMN* edad *INTEGER* (3);

Al usar la función **tiene** se debería que usar las sentencias:

Constructor **tiene** RFC Rfcc

Constructor **tiene** Horario Nombret

Constructor **es** Persona

Con las clases genéricas correspondientes previamente definidas, en donde:

Nombret es un *varchar* de 20 caracteres

Rfcc es un *integer* de 9 caracteres

4.1.4 Implementación de conjuntos

Esta relación consiste en la instancia de una clase a partir de la agrupación de elementos pertenecientes a otra clase. Se asocia esta función con el operador semántico **conjunto**, para la implementación mediante el catálogo de datos, como se muestra en la figura 4.11, se solicitan en el formulario los datos siguientes:

- Nombre de la clase de tipo conjunto. Esta no debe de existir.
- Operador semántico **conjunto**.
- Nombre de la clase. Esta debe existir.

Por ejemplo: **Fraccionamiento** es un conjunto de **casas**. La relación se implementa en el catálogo de datos, como se muestra en la figura 4.11.

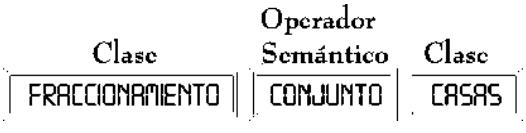


Figura 4.11 Implementación de asociación en el catálogo de datos.

Al crear la clase de tipo conjunto, se implementa una tabla con el nombre de la clase (**fraccionamientos** en el ejemplo), la cual es compuesta por tres campos como se muestra en la figura 4.12:

- La columna de apuntador ID (*key*), la cual se auto incrementa pero permitiendo la existencia de diversos apuntadores iguales. Estos apuntadores iguales indican que pertenecen al mismo grupo.
- Un campo de identificadores para establecer la relación de conjunto con la tabla **fraccionamientos**.
- Una columna denominada identificador el cual guarda el nombre de cada grupo asignado.

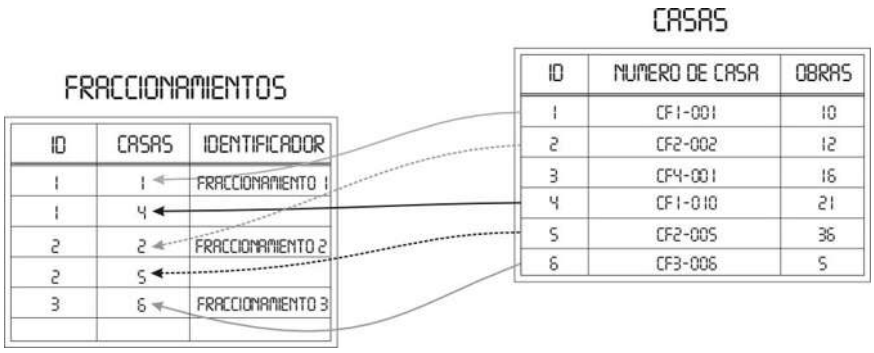


Figura 4.12 Implementación de clase mediante conjuntos.

Se debe señalar que este tipo de clase no puede ser modificada por los métodos de atribución y herencia.

La relación construida debe ser registrada en el índice de relaciones, para facilitar la búsqueda de tablas relacionadas. De acuerdo al tipo de relación se le agrega un indicador, para este caso se le asigna el indicador 1, como se muestra en la figura 4.13.

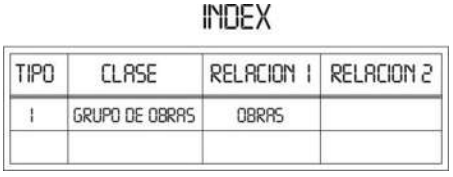


Figura 4.13 Implementación de asociación en el índice.

4.1.5 Implementación de asociación ó tupla

Esta relación consiste en la instancia de una clase que permite la relación de los objetos de dos clases distintas. Asociando esta función con el operador semántico **tupla**. Para la implementación mediante el catálogo de datos, como se muestra en la figura 4.14, se asignan en el formulario los datos siguientes:

- Nombre de la clase. Dado que es la primera clase de la tupla entonces esta clase no debe existir.
- Operador semántico **tupla**.
- Nombre de la primera clase a relacionar, debe existir.
- Nombre de la clase, Debido a que como es la segunda clase de la tupla entonces la clase debe existir.
- Operador semántico **tupla**.
- Nombre de la segunda clase a relacionar, la cual debe existir.

Clase	Operador Semántico	Clase
EXPEDIENTES	TUPLA	DOCUMENTACION
EXPEDIENTES	TUPLA	CONSTRUCTORES

Figura 4.14 Implementación de conjuntos de clases en el catálogo de datos.

Al crear la clase de tipo tupla, se implementa una tabla con el nombre de la clase (**expedientes** en el ejemplo), la cual es compuesta por tres campos como se muestra en la figura 4.15:

- La columna de apuntadores ID (*key*), el cual se incrementa en uno al insertarse un nuevo objeto a la clase. Hay un apuntador único para cada objeto.
- Un campo de apuntadores con el nombre de la primera clase con la cual se relaciona.
- Un campo de apuntadores con el nombre de la segunda clase con la cual se relaciona.

Se debe señalar que este tipo de clase no puede ser modificada por los métodos de atribución y herencia.

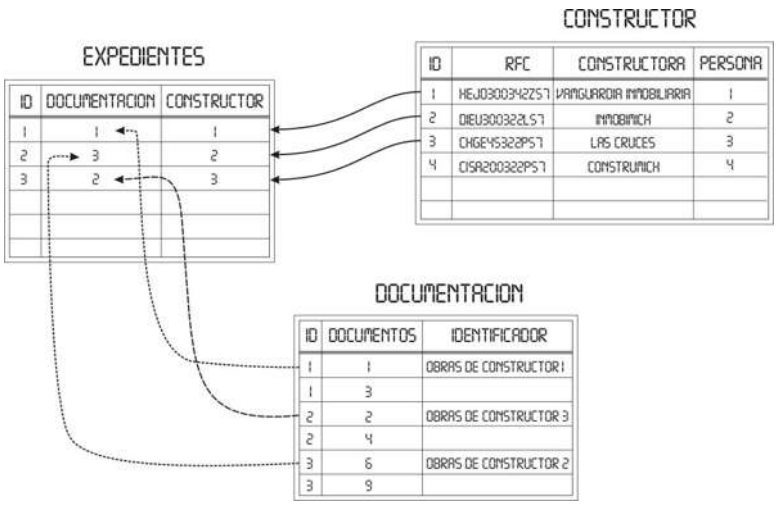


Figura 4.15 Implementación de una clase mediante relación de objetos de dos clases.

La relación construida se debe registrar en el índice de relaciones, para facilitar la búsqueda de tablas relacionadas. De acuerdo al tipo de relación se le agrega un indicador, para este caso se le asigna el indicador 2, como se muestra en la figura 4.16.

INDEX

TIPO	CLASE	RELACION 1	RELACION 2
2	EXPEDIENTES	DOCUMENTACION	CONSTRUCTORES

Figura 4.16 Implementación de tuplas en el índice.

4.2 Consulta de información

La función de búsqueda de información es utilizada para consultar los objetos de las clases. Es implementado de forma que al imprimir una clase, la consulta y extracción de información se realice por ID. Esta función recibe el nombre de la tabla y la ID que se busca, realizando de forma recursiva la siguiente rutina:

1. Si una tabla no tiene relaciones en el índice de relaciones de clases se procede a extraer la información de la manera siguiente:
    - Para aquellos campos que son atributos a partir de clases, se extrae la información del campo y se imprime como un *URL* hacia la misma función con el nombre de la clase y el numero de ID.
    - Para los demás campos se extrae e imprime la información contenida.

2. Si una tabla tiene relaciones en el índice de relación de clases se procede como sigue:
    - Para aquellos campos que son atributos a partir de clases y clases de tipo conjunto o tupla y imprimen como *URL* hacia la misma función con el nombre de la clase y el numero de ID.
    - Si el campo es incluido en el índice de relaciones se reenvía a la función el numero de ID junto con el nombre de la columna.
    - Para los demás campos se extrae e imprime la información contenida.

4.3 Métodos de creación de formularios

Los métodos para la creación de formularios para la inserción y actualización de objetos en cada clase se pueden dividir en tres grupos:

1. Formularios para clases y subclases
2. Formularios para Conjuntos
3. Formularios para Tuplas

4.3.1 Formularios para clases y subclases

La función para crear formularios para la inserción y actualización de objetos es similar en ambos casos. La diferencia radica en que al actualizar se lee el ID del objeto, de modo que se imprimen los valores actuales en el formulario. La función recibe el nombre de la tabla, realizando de forma recursiva la siguiente rutina:

1. Si la tabla no tiene relaciones en el índice de relaciones de clases se procede a imprimir el formulario de la manera siguiente:
    - Para los campos que no son atributos a partir de clases se imprime un menú de selección con las ID existentes en la clase.
    - Para los demás campos se imprime un campo de texto limitado por la cantidad máxima de caracteres a excepción de los campos de tipo *enum*, en el cual se coloca un menú de selección con los valores que puede contener.
2. Si una tabla tiene relaciones en el de relación de clases se procede como sigue:
    - Para aquellos campos que son atributos a partir de clases y clases de tipo conjunto o tupla se imprime un menú de selección con todas las ID existentes.

- Si el campo esta incluido en el índice de relaciones se reenvía a la función.
- Para los demás campos se imprime un campo de texto limitado por la cantidad máxima de caracteres a excepción de los campos de tipo *enum*, en el cual se coloca un menú de selección con los valores que puede contener.

Con esto se alcanza la producción de formularios como el mostrado en la figura 4.17, el cual toma del ejemplo base la clase **locales**.

Figura 4.17 Formulario para clases y subclases.

4.3.2 Formularios para conjuntos

Para crear el formulario correspondiente a clases de tipo conjunto, se imprime un campo para guardar el nombre del grupo que se crea. A continuación se imprime la clase de la cual se seleccionan los datos, como se muestra en la figura 4.18. La información se extrae utilizando la función de consulta iniciando cada ID acompañada de una casilla de selección. En el caso de conjuntos se imprime una casilla por cada uno.

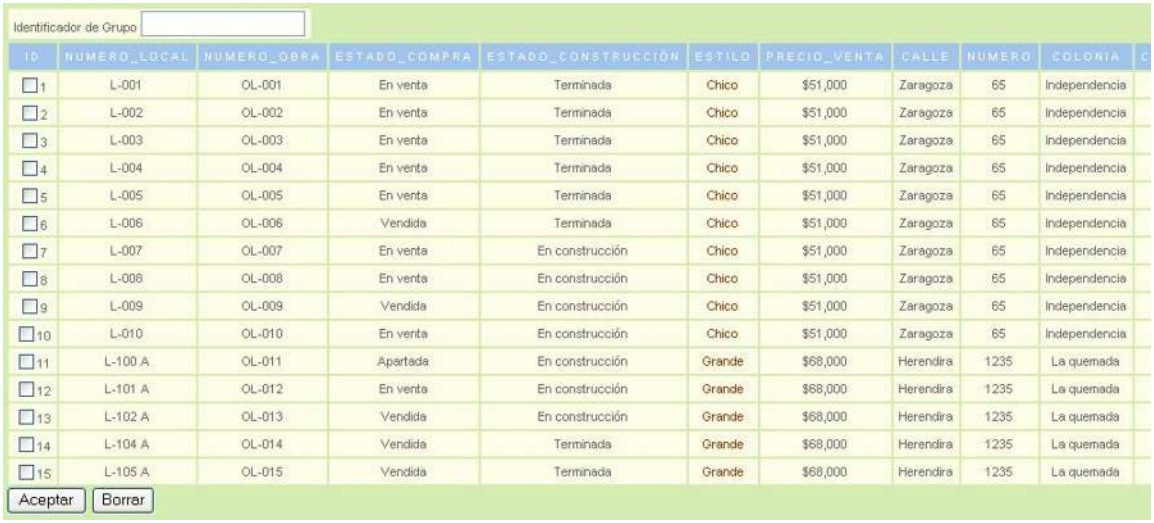


Figura 4.18 Formulario para conjunto.

4.3.3 Formularios para tuplas

Para crear el formulario correspondiente a clases de tipo tupla, se imprimen las dos clases de las cuales se relacionan los datos. La información se extrae utilizando la función de consulta iniciando cada ID acompañada de un botón de opción, como se muestra en la figura 4.19. En el caso de conjuntos se imprime una casilla por cada uno.

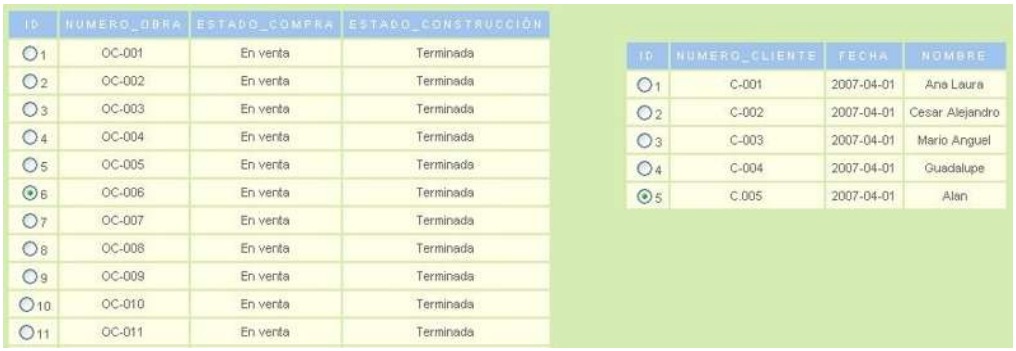


Figura 4.19. Formulario para tuplas.

4.4 Métodos de eliminación de objetos de clases

Para eliminar datos de una clase, se imprime una tabla con todos los renglones iniciando cada ID acompañada de una casilla de selección. En el caso de conjuntos se imprime una casilla por cada uno. Se permite la selección de múltiples renglones y se elimina solamente los renglones de las ID seleccionadas pertenecientes a la tabla.

4.5 Métodos de inserción y actualización de objetos de clases

La función para la inserción y actualización de objetos de las clases recibe el nombre de la tabla y una ID que se crea en caso de insertar información ó se recibe en caso de actualizar, mientras que la información se extrae de una cadena de datos, aplicando de forma recursiva la siguiente rutina:

1. Si una tabla no tiene relaciones en el índice de relaciones de clases se procede a guardar la información de la manera siguiente:
    - Para aquellos campos que son atributos a partir de clases se guarda la información de la cadena de datos, al mismo tiempo que se elimina de la cadena.
    - Para los demás campos se guarda la información extraída de la cadena de datos en el renglón de la ID, al mismo tiempo que se elimina de la cadena.
2. Si una tabla tiene relaciones en el índice de relación de clases se procede como sigue:
    - Aquellos campos que son atributos a partir de clases y clases de tipo conjunto o tupla se guarda el numero de ID de la cadena de datos.
    - Si el campo es incluido en el índice de relaciones se reenvía a la función el nombre de la columna junto con el numero de ID, en caso de ser una actualización de datos. En caso de ser una inserción de datos se reenvía solamente el nombre de la columna.
    - Para los demás campos se guarda la información extraída de la cadena de datos, al mismo tiempo que se elimina de la cadena.

Capítulo 5

Pruebas del sistema

"*Pronto te darás cuenta que hay una gran diferencia entre conocer el camino y caminar el camino*"
Matriz – Morpheus hablando con Neo

"*Nada es demasiado maravilloso para ser verdad*"
Michael Faraday

Capítulo 5

Pruebas del sistema

En el presente capítulo se presenta la implementación del problema base, que fue expuesto en el capítulo 1. Se presenta el planteamiento, diseño e implementación del problema mediante su semántica, presentando de forma esquemática los resultados.

5.1 Planteamiento del problema

El planteamiento del problema representa una de las mayores dificultades al momento de diseñar una base de datos. Es posible que un problema sea planteado de diversas maneras, lo cual generaría una diversa gama de esquemas que de manera significativa satisfagan los requerimientos del problema.

Por lo cual, siendo este un sistema orientado a objetos mediante relaciones semánticas, se debe dividir el problema en la mayor cantidad de clases básicas y definir de forma clara los atributos que deberá contener. A partir de las clases bases también conocidas como superclases se especifican las subclases. Esto se hace de forma que las superclases hereden la mayor cantidad de atributos comunes a las subclases.

Del problema base, indicado en la sección 1.4 del capítulo 1, se puede observar tres distintos tipos de clases:

1. **Personas** que son los constructores y los clientes
2. **Obras** en venta que son casas, departamentos y locales.
3. **Documentación** de cada obra.

La relación de documentos por constructor se puede realizar mediante una tupla denominada: **Expedientes**.

Para ordenar las obras en casas por fraccionamientos, locales por centros comerciales y departamentos por edificios se pueden agrupar las obras desde las subclases: **casas**, **departamentos** y **locales**. Así como se pueden agrupar las obras vendidas y asignar a cada

conjunto el nombre del cliente comprador; otro conjunto se puede crear para las obras apartadas. Entonces se pueden crear los siguientes conjuntos

1. Fraccionamientos.
2. Centros Comerciales.
3. Edificios.
4. Obras compradas.
5. Obras apartadas.
6. Documentación.
7. Grupo de obras.

Con la información deducida se puede generar un esquema lo suficientemente definido para ser implementado mediante el sistema genérico (SGUDEH). El esquema del problema y sus relaciones se muestra en la figura 5.1.

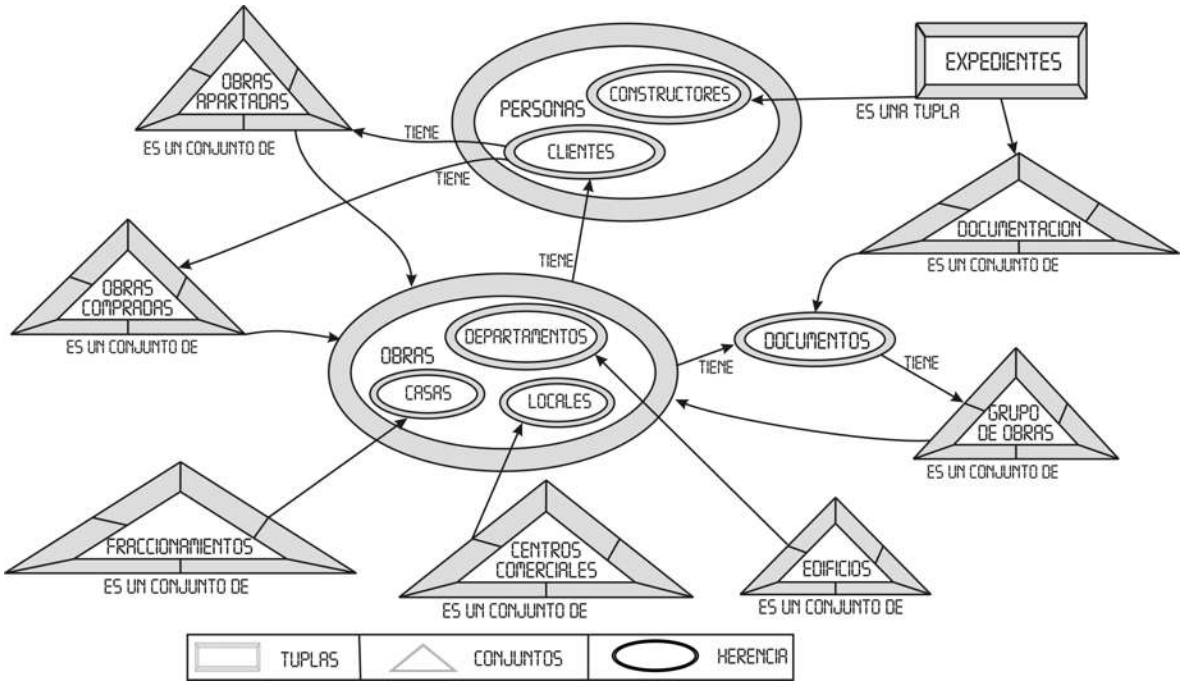


Figura 5.1 Planteamiento del problema.

El siguiente paso es la determinación de los atributos con los que contará cada clase, así como sus respectivas clases genéricas que deberán ser implementadas. Para la clase **personas** y sus subclases **clientes** y **constructores**, se determinan los atributos mostrados en la tabla 5.1.

Tabla 5.1 Atributos de clase persona y subclases constructores y clientes.

Subclase: Clientes	Subclase: Constructores
• No. Cliente. • Fecha.	• RFC. • Compañía. • Tipo persona: moral, física.
Personas	
• Nombre. • Apellido paterno. • Apellido materno. • Edad. • Fecha de nacimiento. • Sexo: Hombre, Mujer. • Curp. • Teléfono.	• Celular. • Calle. • Número. • Colonia. • Código postal. • Ciudad. • Estado.

Como cada tipo de obra: **casas**, **departamentos** y **locales** tienen un estilo de construcción y para evitar la repetición de datos se puede crear una clase de **estilos** la cual será referenciada como atributo a la clase **obra**, de manera que los atributos determinados se muestran en la tabla 5.2.

Tabla 5.2 Atributos de clase estilos.

Clase: Estilos	
• Nombre. • Área. • Número de pisos. • Número de habitaciones.	• Número de baños. • Cocina. • Sala y comedor ó sala/comedor. • Estacionamiento.

La clase **obras** hereda sus atributos a las subclases: **casas**, **departamentos** y **locales**. Los atributos asignados a estas clases son mostrados en la tabla 5.3.

Tabla 5.3 Atributos de clase obras y subclases: casas, departamentos y locales.

Subclase: Casas	Subclase: Departamentos	Subclase: Locales
• Número de casa.	• Número de departamento. • Número de piso: 5 pisos.	• Número de local.
Clase: Obras		
• Número de obra. • Estado de compra. • Estado de construcción. • Precio de venta. • Calle	• Número. • Colonia. • Código postal. • Ciudad. • Estado. • Expediente.	

Los documentos por volumen de obra deben de tener la información mostrada en la tabla 5.4.

Tabla 5.4 Planteamiento del problema.

Clase: Documentos	
• No. Expediente • Fecha de inicio • Fecha de terminación • Costo	• Cantidad • Tipo de obra • Grupo de obras

Una vez determinados los atributos, se deben determinar las clases genéricas que se utilizarán para asignar los tipos nativos a cada atributo. Preferentemente se deben crear de forma que satisfagan la mayor cantidad de atributos sin necesidad de crear una por atributo. Por ejemplo: para atributos como nombres de obras y nombres de usuarios por lo general son claves de tipo M-001 ó RT-001, por lo cual con una clase de tipo *varchar* de 8 caracteres resulta útil para este caso.

En el caso de fechas de nacimiento, de inicio y terminación con una clase de tipo *date* es suficiente.

Para atributos de nombres se puede emplear una clase de tipo *varchar* de 25 caracteres.
En el caso de edades como 25, 75, etc..., con una clase de tipo *varchar* de 3 caracteres es suficiente.
Para atributos de selección como el estado de venta, se puede emplear una clase de tipo *enum* con la cadena: 'En venta', 'Vendida', 'Apartada'. Estos valores se encuentran entre comillas simples (') y separada entre valores por comas (,). Las clases genéricas seleccionadas se muestran en la tabla 5.5.

Tabla 5.5 Clases genéricas.

Clave	Tipo	Longitud/Características
Cantidad	*integer*	2 caracteres
Cel	*integer*	10 caracteres
Claves	*varchar*	8 caracteres
Comedor	*enum*	'cocina/comedor', 'cocina y comedor'
Costo	*varchar*	11 caracteres
CP	*integer*	5 caracteres
Curp	*varchar*	18 caracteres
Edad	*integer*	3 caracteres
EstadoC	*enum*	'Terminada','En construcción'
EstadoV	*enum*	'En venta','Vendida','Apartada'
Estilo	*enum*	'Sencillo','Diamante','Zafiro','Mediterráneo'
Fecha	*date*	
Nombre	*varchar*	25 caracteres
PagosO	*enum*	'Contado', '3 pagos', '6 pagos', '12 pagos'
Persona	*enum*	'Física', 'Moral'
RFC	*varchar*	13 caracteres
Sala	*enum*	'sala/bar','sala y bar'
Sexo	*enum*	'', 'Hombre', 'Mujer'
Telf	*integer*	8 caracteres
YN	*enum*	'Si', 'No'

5.2 Implementación del problema

El procedimiento para la implementación de una base de datos orientada a objetos, consisten en:

1. Creación de la base de datos.
2. Creación de las clases genéricas.
3. Creación de los atributos mediante una clase genérica.
4. Creación de clases de tipo conjunto y tipo tupla
5. Creación de atributos utilizando una clase.
6. Herencia de clases.

5.2.1 Creación de la base de datos

Las bases de datos se crean utilizando las herramientas de administración de bases de datos del sistema, detalladas en la sección 6.1. Después de crearla se debe seleccionar para poder trabajar con ella. La base de datos con la cual se implementara el problema se llama: **constructora**, como se muestra en la figura 5.2.

Figura 5.2 Creación de una base de datos.

5.2.2 Creación de clases genéricas

Las clases genéricas de la tabla 5.5 se implementan mediante el catálogo de clases genéricas en el cual se introducen un nombre clave, el tipo de objeto y sus características, como se muestra en la figura 5.3.

Figura 5.3 Implementación de clases genéricas.

5.2.3 Creación de clases con atributos mediante clases genéricas

El siguiente paso es la creación de los atributos de clases que requieren clases genéricas de las definidas en la tabla 5.5. Se implementan como se muestra en la figura 5.4, teniendo las siguientes consideraciones:

- El orden en el que se crean los atributos es el orden en el que se mostrarán al consultarse la clase.
- Los atributos no deben contener espacios esto es debido a que *MySQL* no los acepta y no se deben crear clases con nombres de atributos.
- El primer campo es el que se muestra como enlace, cuando la clase es utilizada como atributo.

Figura 5.4 Implementación de atributos con clases genéricas.

5.2.4 Creación de clases conjunto y tupla

El siguiente paso consiste en la creación de los conjuntos y tuplas a partir de las clases existentes. Se debe observar que las clases de las cuales se extrae la información deben existir previamente. Por ejemplo, la clase **fraccionamientos** es un conjunto de **casas**; la primera no se puede crear a menos que exista la clase **casas**. Las clases de tipo conjunto y tupla se implementan en el sistema como se observa en la figura 5.5.

Figura 5.5 Implementación de conjuntos y tuplas.

5.2.5 Asignación de atributos utilizando una clase

El siguiente paso consiste en la asignación de atributos que son otras clases incluyendo conjuntos y tuplas. Por ejemplo una **obra** tiene un **expediente** que no se puede asignar a menos que exista. Este tipo de relaciones se implementan en el sistema como se observa en la figura 5.6.

Figura 5.6 Implementación de atributos a partir de clases.

5.2.6 Herencia de clases

Las herencias son el último paso, debido a que el primer campo es utilizado para crear enlaces, en caso de ser una clase asignada como atributo. La herencias de clases son implementas como se muestra en la figura 5.7.

Figura 5.7 Implementación de herencia.

5.3 Edición de clases

La inserción de la información en cada clase se realiza como se indica en la sección 6.3.1.1. Se debe señalar que cada clase tiene un formulario diferente, dependiendo de su tipo. Por ejemplo, al insertar información para la tupla **expedientes** se obtiene el formulario mostrado en la figura 5.8. En donde la información a relacionar se selecciona de las dos tablas.

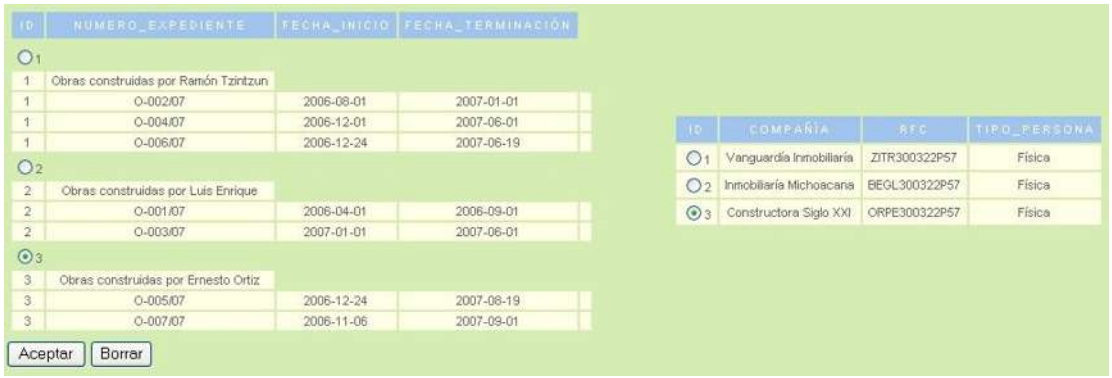
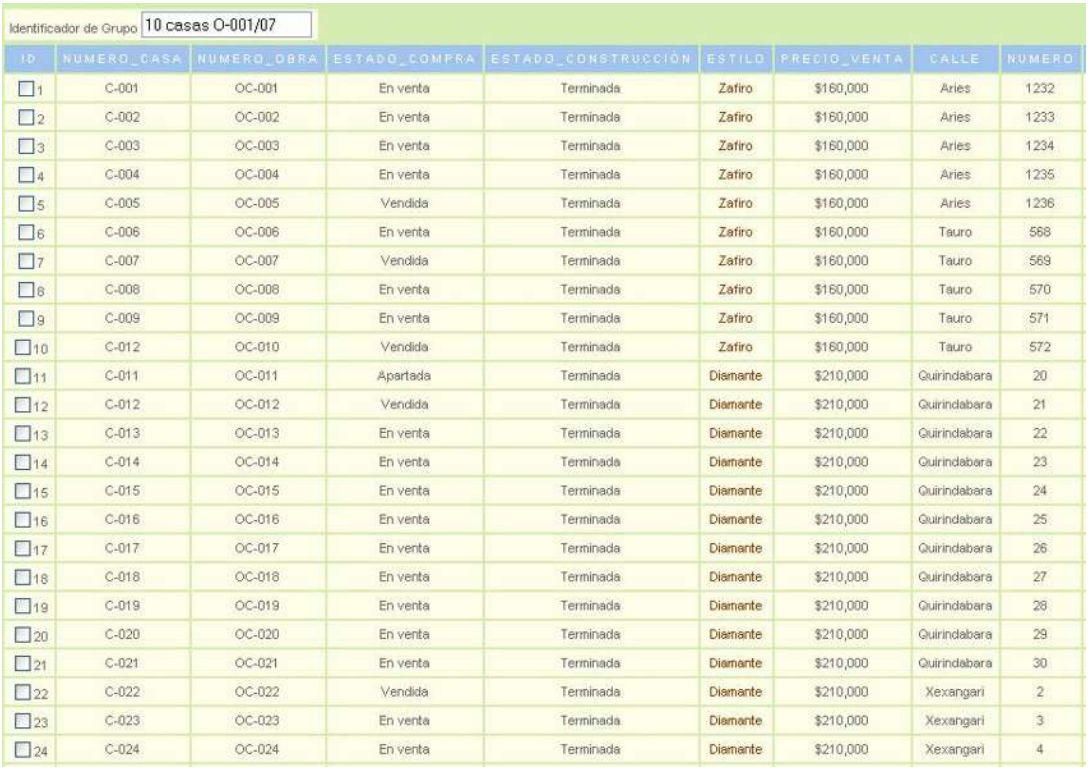


Figura 5.8 Formulario para la inserción de datos de la clase de tipo tupla Expedientes.

Para las clases relacionales como las clases **clientes** y **constructores** se obtienen formularios como los mostrados en la figura 5.9-A y 5.9-B respectivamente.

Figura 5.9 A) Formulario para la inserción de datos de la clase: clientes (derecha). B) Formulario para la inserción de datos de la clase: constructores (izquierda).

Las clases de tipo conjunto como el conjunto **grupo de obras**, tienen un formulario como el mostrado en la figura 5.10. De este se seleccionan las filas de datos que se desean agrupar y se les asigna un nombre como identificador en el campo identificador de conjunto. Este nombre se usa para crear los enlaces hacia esta clase.



Identificador de Grupo: 10 casas O-001/07

ID	NUMERO_CASA	NUMERO_OBRA	ESTADO_COMPRA	ESTADO_CONSTRUCCIÓN	ESTILO	PRECIO_VENTA	CALLE	NUMERO
1	C-001	OC-001	En venta	Terminada	Zafiro	$160,000	Aries	1232
2	C-002	OC-002	En venta	Terminada	Zafiro	$160,000	Aries	1233
3	C-003	OC-003	En venta	Terminada	Zafiro	$160,000	Aries	1234
4	C-004	OC-004	En venta	Terminada	Zafiro	$160,000	Aries	1235
5	C-005	OC-005	Vendida	Terminada	Zafiro	$160,000	Aries	1236
6	C-006	OC-006	En venta	Terminada	Zafiro	$160,000	Tauro	568
7	C-007	OC-007	Vendida	Terminada	Zafiro	$160,000	Tauro	569
8	C-008	OC-008	En venta	Terminada	Zafiro	$160,000	Tauro	570
9	C-009	OC-009	En venta	Terminada	Zafiro	$160,000	Tauro	571
10	C-012	OC-010	Vendida	Terminada	Zafiro	$160,000	Tauro	572
11	C-011	OC-011	Apartada	Terminada	Diamante	$210,000	Quirindabara	20
12	C-012	OC-012	Vendida	Terminada	Diamante	$210,000	Quirindabara	21
13	C-013	OC-013	En venta	Terminada	Diamante	$210,000	Quirindabara	22
14	C-014	OC-014	En venta	Terminada	Diamante	$210,000	Quirindabara	23
15	C-015	OC-015	En venta	Terminada	Diamante	$210,000	Quirindabara	24
16	C-016	OC-016	En venta	Terminada	Diamante	$210,000	Quirindabara	25
17	C-017	OC-017	En venta	Terminada	Diamante	$210,000	Quirindabara	26
18	C-018	OC-018	En venta	Terminada	Diamante	$210,000	Quirindabara	27
19	C-019	OC-019	En venta	Terminada	Diamante	$210,000	Quirindabara	28
20	C-020	OC-020	En venta	Terminada	Diamante	$210,000	Quirindabara	29
21	C-021	OC-021	En venta	Terminada	Diamante	$210,000	Quirindabara	30
22	C-022	OC-022	Vendida	Terminada	Diamante	$210,000	Xexangari	2
23	C-023	OC-023	En venta	Terminada	Diamante	$210,000	Xexangari	3
24	C-024	OC-024	En venta	Terminada	Diamante	$210,000	Xexangari	4

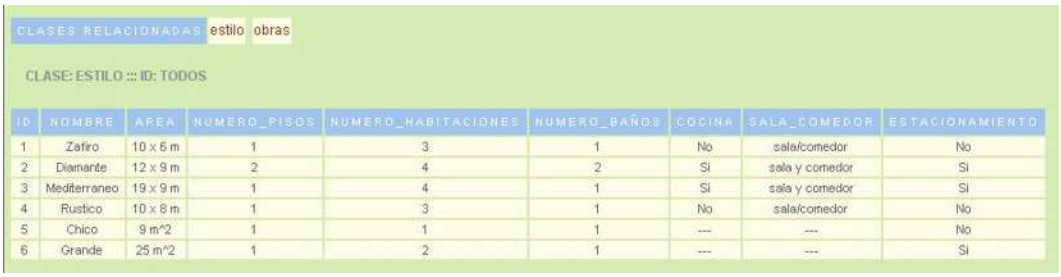
Figura 5.10 Formulario para la inserción de datos del conjunto: grupo de obras.

5.4 Consulta de clases

Una vez guardada la información correspondiente a cada clase, al realizar consultas de información se producen dos diferentes tipos de tablas de información:

- Común para todas las clases y tuplas.
- Una para clases de tipo conjunto

Por ejemplo, las clases relacionales y tuplas se presentan en una tabla como la mostrada en la figura 5.11 que imprime la clase **estilo**.



CLASES RELACIONADAS: estilo obras

CLASE: ESTILO ::: ID: TODOS

ID	NOMBRE	AREA	NUMERO_PISOS	NUMERO_HABITACIONES	NUMERO_BAÑOS	COCINA	SALA_COMEDOR	ESTACIONAMIENTO
1	Zafiro	10 x 6 m	1	3	1	No	sala/comedor	No
2	Diamante	12 x 9 m	2	4	2	Si	sala y comedor	Si
3	Mediterraneo	19 x 9 m	1	4	1	Si	sala y comedor	Si
4	Rustico	10 x 8 m	1	3	1	No	sala/comedor	No
5	Chico	9 m^2	1	1	1	---	---	No
6	Grande	25 m^2	1	2	1	---	---	Si

Figura 5.11 Consulta de la información de la clase: Estilo.

En las clases de tipo conjunto tales como **fraccionamientos**, **centros comerciales** y **edificios** se muestra la información agrupada, cada grupo iniciando con el nombre como se muestra en la figura 5.12.

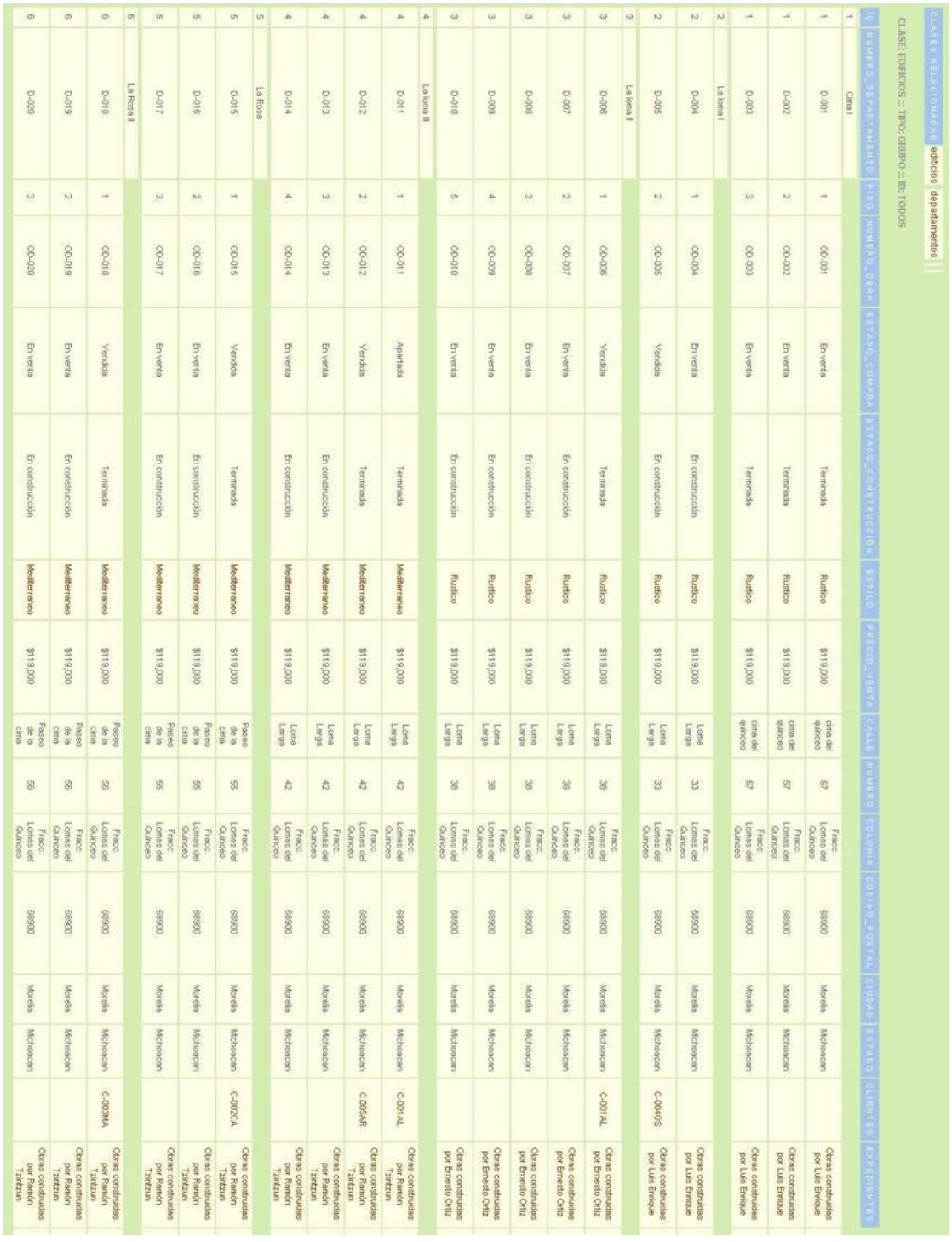
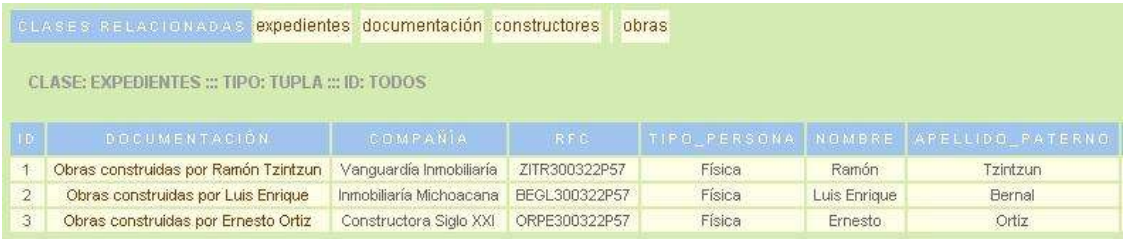


Figura 5.12 Consulta de la información de la clase de tipo conjunto: departamentos.

Se debe señalar que las clases disponen de enlaces, en la parte superior de la tabla de datos denominadas clases relacionadas, los cuales facilitan la navegación entre clases ya que son las clases con las cuales se encuentran relacionadas en forma de atributos o relaciones de herencia. Por ejemplo, la clase de tipo tupla **expedientes** se encuentra relacionada con la clase **documentación** y **constructores**. Así mismo, se encuentra incluida como atributo en la clase **obras**, lo cual se muestra en la figura 5.13.



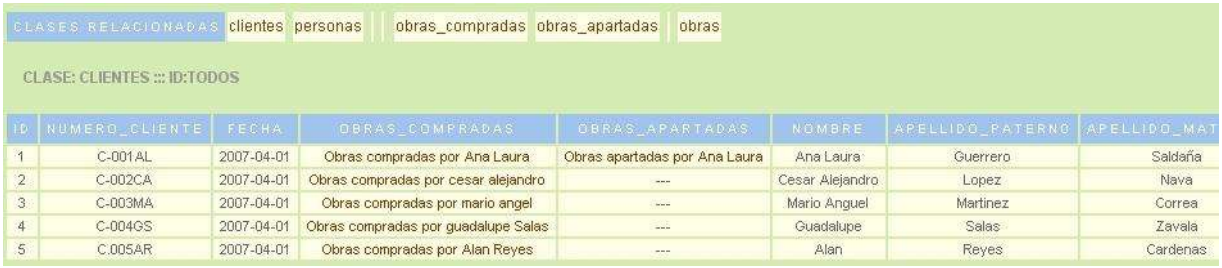
CLASES RELACIONADAS | expedientes | documentación | constructores | obras

CLASE: EXPEDIENTES ::: TIPO: TUPLA ::: ID: TODOS

ID	DOCUMENTACIÓN	COMPAÑÍA	RFC	TIPO_PERSONA	NOMBRE	APELLIDO_PATERNO
1	Obras construidas por Ramón Tzintzun	Vanguardía Inmobiliaría	ZITR300322P57	Física	Ramón	Tzintzun
2	Obras construidas por Luis Enrique	Inmobiliaría Michoacana	BEGL300322P57	Física	Luis Enrique	Bernal
3	Obras construidas por Ernesto Ortiz	Constructora Siglo XXI	ORPE300322P57	Física	Ernesto	Ortiz

Figura 5.13 Consulta de la información de la clase de tipo tupla: Expedientes.

En el caso de la clase **clientes**, los enlaces son las relaciones de herencia que mantiene con personas, así como los atributos que son las clases: **obras compradas** y **obras apartadas**. También, se incluye la clase **obras** la cual tiene como atributo a la clase **clientes**, como se muestra en la figura 5.14.



CLASES RELACIONADAS | clientes | personas | obras_compradas | obras_apartadas | obras

CLASE: CLIENTES ::: ID:TODOS

ID	NUMERO_CLIENTE	FECHA	OBRAS_COMPRADAS	OBRAS_APARTADAS	NOMBRE	APELLIDO_PATERNO	APELLIDO_MAT
1	C-001AL	2007-04-01	Obras compradas por Ana Laura	Obras apartadas por Ana Laura	Ana Laura	Guerrero	Saldaña
2	C-002CA	2007-04-01	Obras compradas por cesar alejandro	---	Cesar Alejandro	Lopez	Nava
3	C-003MA	2007-04-01	Obras compradas por mario angel	---	Mario Anguel	Martinez	Correa
4	C-004GS	2007-04-01	Obras compradas por guadalupe Salas	---	Guadalupe	Salas	Zavala
5	C.005AR	2007-04-01	Obras compradas por Alan Reyes	---	Alan	Reyes	Cardenas

Figura 5.14. Consulta de la información de la clase: Clientes.

Capítulo 6

Documentación

"*Nada Puede tener valor si no es un objeto de utilidad*"
Karl Max

"*…El usuario deberá sentir que controla la computadora, no al revés. Esto se logra con aplicaciones que encarnan tres cualidades: sensibilidad, permisividad y consistencia*."
Inside Macintosh, Volume 1 Apple Computer, Inc. 1985

Capítulo 6

Documentación

En este capítulo se presenta la documentación referente al uso del sistema genérico (SGUDEH). Se realiza énfasis en las interfases *DDL*, *DML* y *DCL* presentándose de forma esquemática las opciones implementadas y señalando de igual manera la sintaxis correcta y el uso apropiado y eficiente de los campos de cada formulario del sistema.

6.1 Interfaz DDL

El sistema genérico (SGUDEH) tiene implementada una interfaz *DDL* la que permite las tres funciones siguientes:

1. Administración de clases genéricas
2. Administración de clases.
3. Eliminación de atributos y clases.

Estas funciones se encuentran divididas en tres secciones dentro de la sección catálogo como se muestra en la figura 6.1.

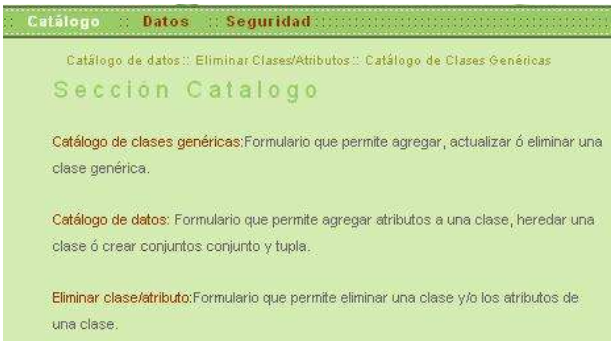


Figura 6.1 Sección catálogo.

6.1.1 Administración de clases genéricas

En la sección **catálogo de clases genéricas** se encuentra el formulario para la administración de clases genéricas mostrado en la figura 6.2. Este formulario permite tres funciones:

1. Crear una clase genérica
2. Actualizar una clase genérica
3. Eliminar una clase genérica

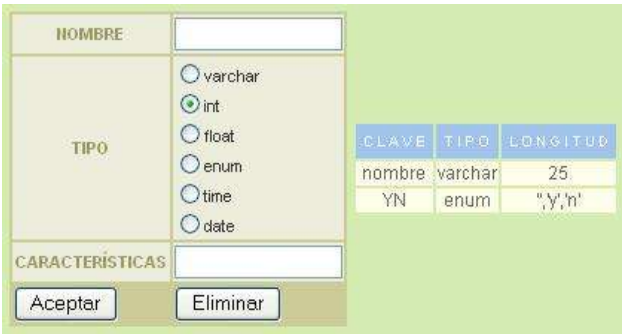


Figura 6.2 Catálogo de clases genéricas.

Una clase genérica esta compuesta de tres partes:

- Nombre.
- Tipo nativo.
- Características.

El sistema tiene seis tipos nativos implementados:

- *varchar.*
- *int.*
- *float.*
- *enum.*
- *time.*
- *date.*

Crear una clase genérica

Para crear una clase genérica de los tipos *varchar*, *int* y *float*, se introduce un número entero en el campo **características** que representa la cantidad de caracteres que tendrá.
Para crear una clase genérica del tipo *enum* se introduce en el campo **características** los valores entre comillas simples y separadas por comas. Por ejemplo si se desea que un campo tenga los valores **si** y **no** se introduce como sigue: 'Si', 'No'.
Para crear una clase genérica del tipo *time* ó *date* no se introduce ningún valor.
Una vez introducidos los valores se presiona **Aceptar** para guardar.

Actualizar una clase genérica

Para actualizar los datos de una clase genérica, en el campo **nombre** se introduce el nombre de la clase genérica que se desea actualizar. Se insertan los valores en los campos **tipo** y **características** como se describe en la sección anterior. Para guardar los cambios se presiona **Aceptar**.

Se debe señalar que al actualizar una clase genérica después de haber sido empleada en la creación de un atributo de alguna clase, las características del atributo no se verán afectadas.

Eliminar una clase genérica

Para eliminar una clase genérica se debe introducir su nombre en el campo **nombre** y se presiona la opción **Eliminar**.

Se debe tener en cuenta que al eliminar una clase genérica después de haberla usado en la creación de algún atributo de una clase, el atributo no será afectado.

6.1.2 Administración de clases

La interfaz para describir clases y atributos se encuentra en la sección **Catálogo de datos**, en esta sección se cuenta con un formulario como el mostrado en la figura 6.3, con el cual se puede realizar las siguientes funciones:

1. Atribución.
2. Herencia.
3. Conjunto.
4. Tupla.

Figura 6.3 Catálogo de datos.

Al utilizar el formulario se deben tener las siguientes precauciones:

- Se debe señalar que el sistema no acepta ningún nombre que contenga espacios y su uso ocasionará un error en el sistema, esto es debido a que Mysql no soporta tablas y nombre de campos que contengan espacios.
- Están permitidos el uso de vocales mayúsculas y minúsculas con acentos (´) y diéresis (¨), el uso de eñe mayúscula (Ñ) y minúscula (ñ).
- Esta permitido el uso del símbolo (_) en nombres de clases o atributos.
- Otros símbolos diferentes a los alfabéticos no están permitidos (¡, ”, ·, $, %, &, etc...) y su uso causará un error en el sistema.
- Finalmente se debe señalar que hay palabras y nombres que no es posible utilizar debido a que son palabras reservadas; estas se encuentran indicadas en la tabla 6.1.

Tabla 6.1 Palabras reservadas.

tiene	es	*index_atrib*	*index_tablas*
conjunto	tupla	*Profiles*	*my_definitions*

Atribución

El operador semántico utilizado para esta función es **tiene**. No se puede aplicar la atribución en clases de tipo: conjunto o tupla. Se pueden presentar dos casos de atribución: utilizando una clase genérica ó una clase existente.

Para asignar un atributo de una clase genérica, la cual debe de existir previamente, se utiliza la sintaxis ejemplificada en la figura 6.4, que debe tener la forma:

Nombre de clase + **tiene** + Nombre de atributo + Definición

Figura 6.4 Sintaxis de atribución empleando una clase genérica.

Para el caso de atribución de una clase existente, ejemplificada en la figura 6.5, la sintaxis debe tener la forma:

Nombre de clase+ **tiene** + Nombre de clase (existente)

Figura 6.5 Sintaxis de atribución empleando una clase.

Herencia

El operador semántico utilizado para esta función es: **es**. Para asignar la herencia entre clases estas deben existir. De manera que la sintaxis, ejemplificada en la figura 6.6, es de la forma:

Nombre de tabla + **es** + Nombre de tabla (existente)

Figura 6.6 Sintaxis de herencia de una clase.

Conjunto

El operador semántico utilizado para esta función es: **conjunto**. Así que la sintaxis, ejemplificada en la figura 6.7, es de la forma:

Nombre del conjunto + **conjunto** + Nombre de clase (existente)

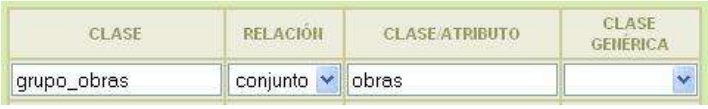


Figura 6.7 Sintaxis de conjunto.

Tupla

El operador semántico utilizado para esta función es: **tupla**. Siendo la sintaxis, ejemplificada en la figura 6.8, de la forma:

Nombre de la tupla + **tupla** + Nombre de la primera clase (existente)

Nombre de la tupla + **tupla** + Nombre de la segunda clase (existente)

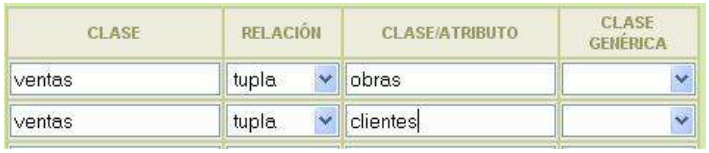


Figura 6.8 Sintaxis de tupla.

6.1.3 Eliminación de atributos y clases

En la sección **Eliminar clases/atributos** se encuentra un formulario como el mostrado en la figura 6.9, el cual realiza las siguientes dos funciones:

1. Eliminar una clase
2. Eliminar un atributo de una clase

Figura 6.9 Formulario para eliminación de clases y atributos.

Eliminar una clase

Es posible eliminar una o varias clases a la vez, para lo cual se deben seleccionar del campo **clases** aquellas a eliminar y seleccionar la opción clase del campo ***DDL*** mostrados en la

figura 6.9. Para continuar se presiona **eliminar**, después de lo cual se deberá aceptar una advertencia para borrar las clases.

Se deben tener en consideración las siguientes precauciones:

- Al eliminar cualquier clase se perderá toda la información contenida.
- Las relaciones de grupos, tuplas o dependientes de las características de otras tablas se podrían alterar y ocasionar errores.
- Debe verificarse como se alteran las relaciones entre clases antes de realizar cualquier cambio.

Eliminar un atributo de una clase

Se puede eliminar el atributo de una clase utilizando el formulario de la figura 6.9, seleccionando la clase a modificar y presionando **Eliminar** para continuar. Después se generará un formulario como el mostrado en la figura 6.10, del cual se podrá eliminar un solo atributo por vez, se selecciona el nombre del atributo y se presiona **eliminar atributo** para continuar.

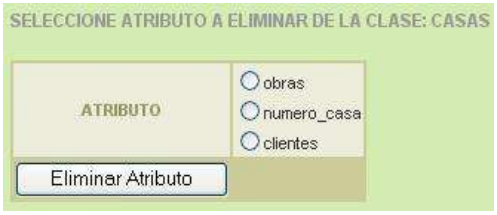


Figura 6.10 Ejemplo eliminando campos de una tabla.

Para finalizar se confirma la advertencia que se genera en caso de desear eliminar dicho atributo. Se deben tener en cuenta las siguientes consideraciones:

- Se podrán eliminar toda clase de campos aún conteniendo información, por lo cual se perderá toda la información contenida.
- Debe verificarse como se alteran las tablas antes de realizar cualquier cambio.

6.2 Interfaz DML

El sistema genérico (SGUDEH) cuenta con una interfaz *DML* que realiza las dos funciones siguientes:

1. Edición de datos de una clase.
2. Consultar de datos de una clase

Las cuales se encuentran en la sección **Datos**, divididas en dos como se muestra en la figura 6.11.

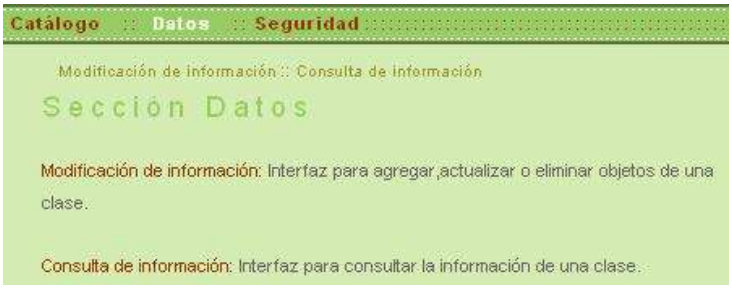


Figura 6.11 Sección datos.

6.2.1 Edición de datos de una clase

Para editar los datos de una clase se recurre a las herramientas que se encuentran en la sección **Modificación de información**, lo cual presenta un formulario como el mostrado en la figura 6.12.

Figura 6.12 Formulario para la edición de datos.

Las funciones implementadas para la edición de datos son las siguientes:

1. Insertar datos.
2. Actualizar datos.
3. Eliminar datos.

1. Insertar datos

Para insertar datos a una clase, conjunto o tupla, del formulario de la figura 6.12 se realiza el siguiente procedimiento:

- Seleccionar la clase del menú desplegable **clase**.
- Seleccionar la opción **insertar** del campo *DML*.
- Presionar **Editar**.

Esto genera tres tipos de formularios básicos para insertar datos:

1. Formularios para clases.
2. Formularios para conjuntos.
3. Formularios para tuplas.

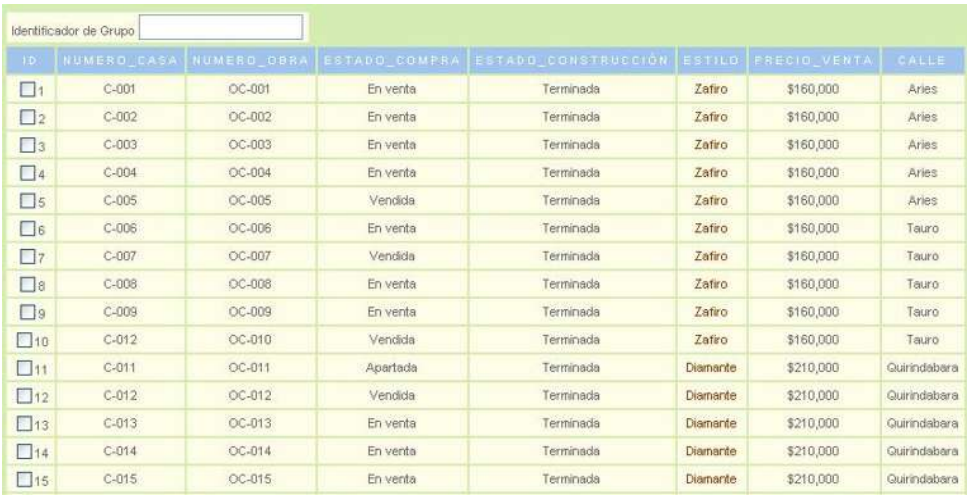
Formularios para clases

Los formularios de clases son generados con los campos de la clase, como el mostrado en la figura 6.13, el cual deberá llenarse con la información correspondiente. Para volver a los valores por defecto se presionar **Borrar**. Para guardar la información se elige **Aceptar**.

Figura 6.13 Formulario para la edición de datos de clases.

Insertar información a conjuntos

Los formularios de conjuntos como el mostrado en la figura 6.14 imprimen las clases de la cual se crearan los conjuntos, con una casilla seleccionable por ID. Se incluye un campo denominado **Identificador de grupo**, al cual se le asigna un nombre por conjunto.



Identificador de Grupo

ID	NUMERO_CASA	NUMERO_OBRA	ESTADO_COMPRA	ESTADO_CONSTRUCCIÓN	ESTILO	PRECIO_VENTA	CALLE
1	C-001	OC-001	En venta	Terminada	Zafiro	$160,000	Aries
2	C-002	OC-002	En venta	Terminada	Zafiro	$160,000	Aries
3	C-003	OC-003	En venta	Terminada	Zafiro	$160,000	Aries
4	C-004	OC-004	En venta	Terminada	Zafiro	$160,000	Aries
5	C-005	OC-005	Vendida	Terminada	Zafiro	$160,000	Aries
6	C-006	OC-006	En venta	Terminada	Zafiro	$160,000	Tauro
7	C-007	OC-007	Vendida	Terminada	Zafiro	$160,000	Tauro
8	C-008	OC-008	En venta	Terminada	Zafiro	$160,000	Tauro
9	C-009	OC-009	En venta	Terminada	Zafiro	$160,000	Tauro
10	C-012	OC-010	Vendida	Terminada	Zafiro	$160,000	Tauro
11	C-011	OC-011	Apartada	Terminada	Diamante	$210,000	Quirindabara
12	C-012	OC-012	Vendida	Terminada	Diamante	$210,000	Quirindabara
13	C-013	OC-013	En venta	Terminada	Diamante	$210,000	Quirindabara
14	C-014	OC-014	En venta	Terminada	Diamante	$210,000	Quirindabara
15	C-015	OC-015	En venta	Terminada	Diamante	$210,000	Quirindabara

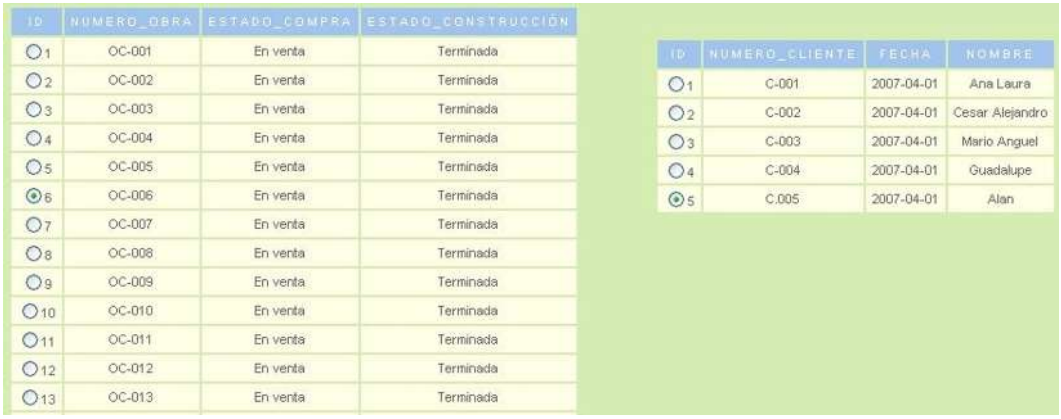
Figura 6.14 Formulario para la edición de datos conjuntos.

Para crear un conjunto se debe seleccionar las ID a agrupar en un conjunto y asignar un nombre. Para volver a los valores por defecto se presiona **borrar**. Para guardar la información se elige **Aceptar**.

Insertar información a tuplas

Los formularios de tuplas imprimen las dos clases de las cuales se relacionaran los datos, como el mostrado en la figura 6.15. De cada clase solo se imprimen los primeros cuatro atributos con una casilla seleccionable por ID.

Para relacionar los datos se debe seleccionar una casilla de ID por clase. Para volver a los valores por defecto se presionar **borrar**. Para guardar la información presionar **Aceptar**.



ID	NUMERO_OBRA	ESTADO_COMPRA	ESTADO_CONSTRUCCIÓN
1	OC-001	En venta	Terminada
2	OC-002	En venta	Terminada
3	OC-003	En venta	Terminada
4	OC-004	En venta	Terminada
5	OC-005	En venta	Terminada
6	OC-006	En venta	Terminada
7	OC-007	En venta	Terminada
8	OC-008	En venta	Terminada
9	OC-009	En venta	Terminada
10	OC-010	En venta	Terminada
11	OC-011	En venta	Terminada
12	OC-012	En venta	Terminada
13	OC-013	En venta	Terminada

ID	NUMERO_CLIENTE	FECHA	NOMBRE
1	C-001	2007-04-01	Ana Laura
2	C-002	2007-04-01	Cesar Alejandro
3	C-003	2007-04-01	Mario Anguel
4	C-004	2007-04-01	Guadalupe
5	C.005	2007-04-01	Alan

Figura 6.15 Formulario para la edición de datos de tuplas.

2. Actualizar datos de una clase

Para actualizar los datos de una clase se debe conocer el ID del renglón del que modificará información. A partir del formulario de la figura 6.12 se deben seguir los siguientes pasos:

- Seleccionar el nombre de la clase del menú desplegable **clase**.
- Seleccionar la opción **actualizar** del campo *DML*.
- Introducir el numero de ID a modificar en el campo **ID**.
- Presionar **Editar**.

Esto genera los tres tipos de formularios básicos mostrados en la sección anterior, con la información actual. Se modifica la información de acuerdo al tipo de formulario al terminar se presiona **aceptar** para guardar los cambios generados.

3. Eliminar datos

Para eliminar los datos de una clase, conjunto o tupla, a partir del formulario de la figura 6.12 se deben seguir los siguientes pasos:

- Seleccionar la clase del menú desplegable **clase**.
- Selecciona la opción **eliminar** del campo *DML*.
- Presionar **Editar**.

Lo cual imprime la información de la clase con un campo seleccionable por ID, como se muestra en la figura 6.16.

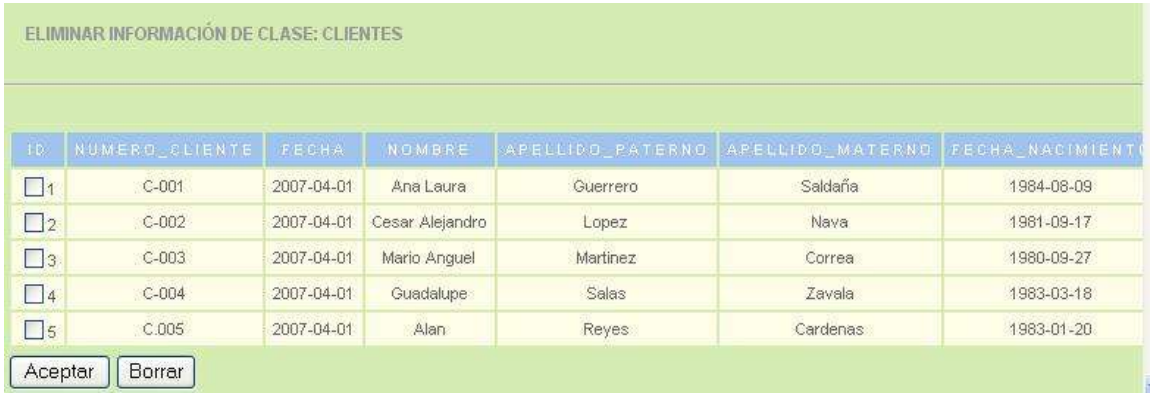


Figura 6.16 Formulario para eliminar datos.

Para eliminar una ó múltiples ID se debe seleccionar de la lista y presionar **Aceptar**. Solamente la información contenida en la tabla será eliminada, la información contenida en otras se mantendrá intacta.

6.2.2 Consulta de datos de una clase

Para consultar la información de una clase se debe ir a la sección **Consulta de información**, mostrada en la figura 6.11. En esta sección se despliega un formulario, como se muestra en la figura 6.17, en el cual se debe seleccionar del campo **clase** el nombre de la clase de la que se desea consultar su información.

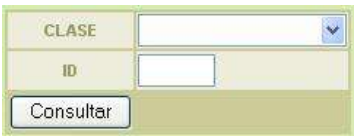
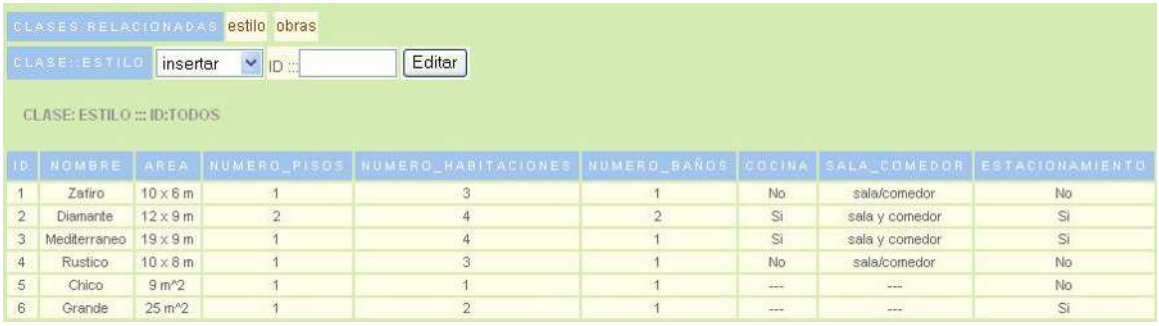


Figura 6.17 Formulario para la edición de datos de clases.

Una solicitud imprime en una tabla la información de la clase, como se muestra en la figura 6.18. Al agregar una ID en el campo **ID**, se imprime únicamente su información.

Cuando se muestra la información de una clase, en la parte superior se cuenta con enlaces que indican las clases con las que se encuentra relacionada, al ser presionados estos permiten solicitar la información de las clases. También se incluye un menú para acceder de manera rápida a las opciones de edición para la clase actual, las cuales funcionan como se describió en la sección 6.2.1.



ID	NOMBRE	AREA	NUMERO_PISOS	NUMERO_HABITACIONES	NUMERO_BAÑOS	COCINA	SALA_COMEDOR	ESTACIONAMIENTO
1	Zafiro	10 x 6 m	1	3	1	No	sala/comedor	No
2	Diamante	12 x 9 m	2	4	2	Si	sala y comedor	Si
3	Mediterraneo	19 x 9 m	1	4	1	Si	sala y comedor	Si
4	Rustico	10 x 8 m	1	3	1	No	sala/comedor	No
5	Chico	9 m^2	1	1	1	---	---	No
6	Grande	25 m^2	1	2	1	---	---	Si

Figura 6.18 Formulario para la edición de datos de objetos.

6.3 Interfaz DCL

El sistema genérico (SGUDEH) cuenta con tres interfaces *DCL* para la protección de datos, mostrados en la figura 6.19, las cuales realizan las siguientes funciones:

1. Administración de usuarios.
2. Administración de bases de datos.
3. Protección mediante archivos htaccess.

Estas funciones son ejecutadas mediante las interfaces de la sección **seguridad**, la cual cuenta con tres subsecciones: **usuarios**, **databases** y **htaccess;** que son configuradas por el administrador ***root***, el cual es el superusuario por defecto de *MySQL,* por lo cual se debe conocer la contraseña actual en caso de existir, para las instalaciones recientes no existe contraseña por defecto.

Figura 6.19 Formulario para la edición de datos de seguridad.

6.3.1 Administración de usuarios

En la subsección **usuarios** de la sección **seguridad**, se encuentran las herramientas para la administración de usuarios. Se imprime una tabla con todos los usuarios existentes creados con Mysql, como se muestra en la figura 6.20, aunque solamente los usuarios administrados con las herramientas del sistema y que pertenecen al *host* local (*localhost*) tienen acceso al sistema.

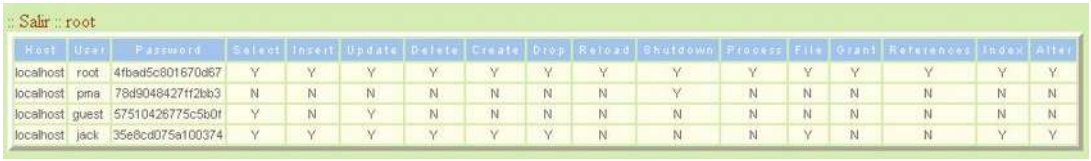


Figura 6.20 Sección administración de usuarios

Los usuarios administrados por el sistema cuentan con una sesión, lo cual permite que varios usuarios estén conectados y trabajando con la base de datos actual. Para iniciar una sesión se debe registrar mediante el formulario de la sección ***login***, el cual acepta datos de nombre de usuario y contraseña, como se muestra en la figura 6.21.

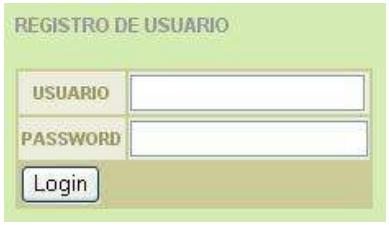


Figura 6.21 Registro de usuario.

Una vez iniciada una sesión, esta podrá ser terminada utilizando el enlace **salir**. Se debe señalar que por cuestiones de seguridad la duración de la sesión es de 3 horas, una vez transcurrido el tiempo de sesión el usuario deberá de registrarse nuevamente en la sección ***login***.

Para administrar usuarios se tienen cuatro funciones, en la sección **usuarios**, que realizan:

1. Crear un nuevo usuario.
2. Actualizar contraseñas (*passwords*).

3. Actualizar permisos de usuarios.
4. Eliminar usuarios

Las cuales son ejecutables mediante las opciones del formulario mostrado en la figura 6.22.

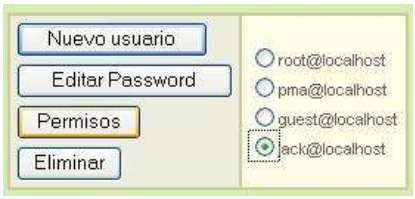


Figura 6.22 Formulario para la administración de usuarios.

Crear un nuevo usuario

Para crear un nuevo usuario debe presionarse **Nuevo usuario**, del formulario mostrado en la figura 6.22, el cual desplegará otro formulario como el mostrado en la figura 6.23. Para crear el usuario debe elegirse **Crear**.

Figura 6.23 Formulario para crear nuevos usuarios.

A los campos se les asigna la siguiente información:

- *Host*: Es el nombre de servidor desde el cual el usuario puede acceder para trabajar con la base de datos puede ser local (*localhost*), un servidor en especifico ó cualquiera. En caso de ser cualquier servidor se utiliza el símbolo (%).
- Usuario: Nombre del nuevo usuario que se desea crear, el cual no puede estar duplicado aún cuando tuvieran *host* diferentes.
- Contraseña: Es la contraseña que se le asignará al usuario. Por seguridad debe ser asignada.

- Privilegios: Son los permisos de Mysql asignados por usuario y señala las operaciones que se podrán realizar a las tablas del sistema. Los privilegios necesarios para la lectura y edición de datos, creación y eliminación de clases, se muestran en la tabla 6.2. El significado de cada permiso se incluye en el apéndice B.

Tabla 6.2 Permisos necesarios para las operaciones de usuarios.

Operación	Privilegios Seleccionados	
Lectura de datos	*Select*	*Update*
Creación y eliminación de clases	*Create* *Drop* *Alter* *Delete*	*Select* *Insert* *Update*
Edición de datos	*Select* *Insert*	*Update* *Delete*

Actualizar contraseñas

Para actualizar la contraseña de un usuario se debe seleccionar el usuario de la lista, mostrada en la figura 6.22, y presionar **Editar** ***Password***. Esto despliega un formulario como el mostrado en la figura 6.24, el cual pide una nueva contraseña que deberá ser reescrito en el campo **Confirmar** ***password***. Para finalizar debe presionarse **Actualizado**.

Figura 6.24 Selección de usuarios.

Actualizar privilegios de usuarios

Para actualizar los privilegios de un usuario se debe seleccionar un usuario de la lista, mostrada en la figura 6.22, y presionar la opción **Permisos**. Esto despliega un formulario con los privilegios actuales del usuario, como se muestra en la figura 6.25, en el cual se

podrá cambiar los permisos de los usuarios actuales. Para guardar los cambios realizados debe presionarse **Actualizar**.

Figura 6.25 Formulario para actualizar permisos.

Eliminar usuarios

Para eliminar algún usuario debe seleccionársele de la lista de usuarios mostrada en la figura 6.22, y presionar **Eliminar**. Esto generara una advertencia como la mostrada en la figura 6.26 que deberá aceptar en caso de desear eliminar el usuario.

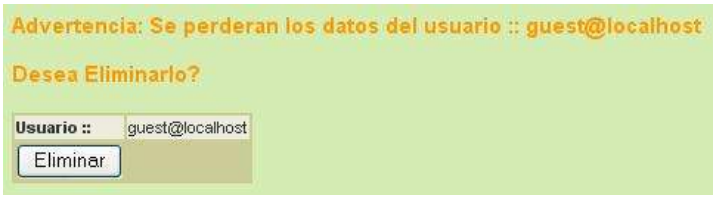


Figura 6.26 Advertencia al eliminar un usuario.

6.3.2 Administrador de bases de datos

En la sección **Databases**, se presenta un formulario mostrado en la figura 6.27, que permite realizar las siguientes funciones para manipular databases:

1. Crear una base de datos.
2. Eliminar una base de datos.
3. Seleccionar una base de datos.

Figura 6.27 Administrador de bases datos.

Las bases de datos creadas se muestran en una lista y se puede seleccionar, una por operación, para las opciones **Eliminar database** y **Seleccionar database**.

Crear una base de datos

Para crear una nueva base de datos se debe realizar el siguiente procedimiento:

- Presionar **Crear database**. Esta opción presenta un formulario como se muestra en la figura 6.28
- En el campo **Database** se introduce el nombre de la base de datos que se desee crear.
- Presionar **crear**.

Si ya existe la base de datos se imprimirá en pantalla un mensaje de error que indicara que ya existe.

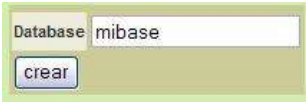


Figura 6.28 Formulario para crear una base de datos.

Al crearse la base de datos se mostrará un mensaje con las tablas por defecto creadas por el sistema. Se debe señalar que solamente las bases de datos creadas por el programa aparecen en la lista de bases.

Eliminar una base de datos

Para eliminar una base de datos debe realizar el siguiente procedimiento:

- Seleccionar la base de datos a eliminar de la lista mostrada en la figura 6.27.
- Presionar **Eliminar database**.

- Aceptar la advertencia que aparece, como se muestra en la figura 6.29.

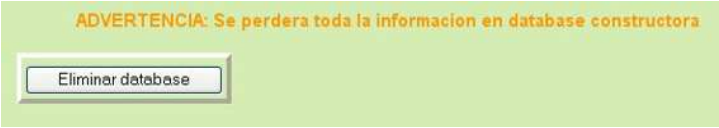


Figura 6.29 Advertencia al eliminar una bases de datos.

Se deben tener en cuenta las siguientes precauciones:

- Al eliminar la base de datos se perderá toda la información que pueda contener.
- Solamente las bases creadas por el sistema son las que se pueden eliminar.

Seleccionar una base de datos

Para seleccionar y utilizar una base de datos existente para que el sistema trabaje con ella se debe realizar el siguiente procedimiento:

- Seleccionar una base de datos de la lista mostrada en la figura 6.27.
- Presionar **seleccionar database**.

La base de datos puede cambiarse cada vez que se use la opción **seleccionar database.** La base actual aparece en la parte inferior del formulario de la figura 6.27.

6.3.3 Protección mediante archivos htaccess

En la sección ***htaccess***, se encuentra una interfaz para proteger el sistema mediante archivos *htaccess*, esta interfaz permite las funciones siguientes:

1. Guardar rutas para archivos *htaccess* y archivos de contraseñas.
2. Generar archivos htaccess.
3. Eliminación de los archivos *htaccess*.

Estas opciones son generadas mediante el formulario que se muestra en la figura 6.30.

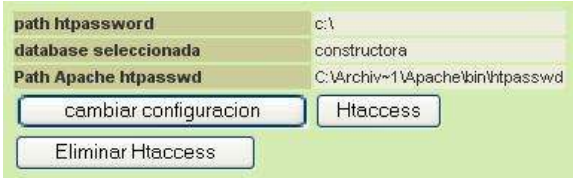


Figura 6.30 Formulario para configurar protección con htaccess.

Guardar rutas para archivos htpasswd y archivos de contraseñas

Antes de usar la protección mediante archivos *htaccess* se deben de guardar dos rutas necesarias para su ejecución:

- La ruta hacia el programa *htpasswd* ó *htpasswd2* en Apache 2.
- La ruta en donde se guardarán los archivos de contraseñas generados por el programa *htpasswd*.

El formulario para guardar las rutas se genera al presionar **cambiar configuración** como se observa en la figura 6.31.

Figura 6.31 Formulario para guardar rutas.

Antes de introducir una ruta en donde se guarden los archivos de contraseñas generados por *htpasswd*, para cualquier sistema se deben tener las siguientes precauciones:

- La carpeta debe existir.
- La ruta de la carpeta no debe ser vista desde *Internet*.
- Guardar la ruta del programa de acuerdo al sistema operativo (*Windows* y *Linux*).

Sistemas *Windows*:

En caso de encontrase registrada en la variable *path* la ruta hacia el programa *htpasswd*, solamente se deberá guardar el nombre del programa (*htpasswd*). Por lo general no se encuentra registrada en la variable *path*, entonces, se deberá guardar la ruta completa en donde se encuentra, el ejecutable generalmente se encuentra en la carpeta bin de Apache (..\Apache\bin).

Debido a que los sistemas *Windows* almacenan las direcciones de carpeta como un nombre de ocho caracteres y una extensión de tres (once caracteres en total), entonces, las carpetas con una longitud de mas de 8 caracteres o con espacios se les deberá agregar después del sexto carácter los caracteres (~1). Por ejemplo sea la ruta donde se encuentra *htpasswd*:

C:\Archivos de programa\Apache\bin\htpasswd

La ruta que se debe guardar es como sigue:

C:\Archiv~1\Apache\bin\htpasswd

Para el caso de la ruta para contraseña se debe tener las siguientes precauciones

- No usar trayectorias con espacios.
- Agregar backslash al final de la dirección (\).

Entonces siendo la ruta donde se guardaran los archivos de contraseña:

C:\Archivos de programa*passwords*

La ruta que se debe guardar es como sigue:

C:\Archiv~1\ *passwords*\

Sistemas Linux:

Por lo general, en estos sistemas operativos se encuentran configurado para que *htpasswd* se ejecute, por lo tanto no es necesario insertar el trayecto del *htpasswd*, solamente se introducirá el nombre del programa que esta escrito por defecto (*htpasswd*). Cuando se tiene instalado Apache2 el nombre del programa cambia a: *htpasswd2*.

Para asignar la dirección de carpeta en sistemas con SO *Linux* se deben tomar en cuenta los siguientes puntos:

- El trayecto a la carpeta se debe introducir completa.
- No usar rutas con espacios.
- Agregar *slash* al final de la dirección (/). Por ejemplo:

/*usr*/*passwords*/

Generar archivos htaccess

Al presionar ***htaccess***, se puede generar un archivo *htaccess* el cual sólo permite el acceso de los usuarios del sistema. De esta manera se realiza la protección del sistema y al iniciar

el programa se genera un cuadro de dialogo del sistema operativo, como el que se muestra en la figura 6.32, que pide usuario y contraseña cada vez que se desee tener acceso al sistema.

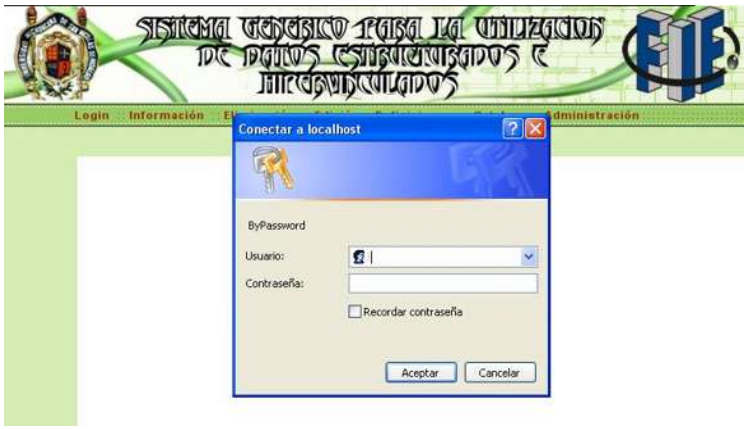


Figura 6.32 Programa protegido por htaccess.

Sólo se deberá de tener las siguientes precauciones antes de usar esta opción:

- Para los usuarios de *Linux* se deberá de contar con permisos de acceso completo (777), en la carpeta donde se generen los archivos *htpassword* y también en las carpetas de administración y de programa del sistema. Una vez realizados los cambios se restablecen los permisos originales (755). Los permisos se otorgan mediante el comando chmod.
- Para los usuarios de *Windows* por lo general se cuenta con acceso total pero puede ser que en algunos casos interfiera el *firewall* o el antivirus por lo que se recomienda desactivarlos antes de realizar la operación.
- Tener registrados los usuarios del sistema y generar la protección nuevamente al agregar o eliminar un nuevo usuario.
- Haber realizado la configuración del *htpasswd* y de la carpeta en donde se crearan los archivos de contraseñas. Esto se realiza en la sección **Configuración**.

Si no se toman en cuenta las advertencias anteriores puede ocurrir que:

- El programa sólo genere el archivo *htaccess* y se haga inaccesible el programa a los usuarios.
- No se genere ningún archivo.

Eliminar la protección htaccess

Al presionar **Eliminar *htaccess*** permite eliminar los archivos generados por *htpasswd* así como los archivos *htaccess*, por lo que se debe tener las siguientes precauciones:

- Para los usuarios de *Linux* se deberá de contar con permisos de acceso completo (777), en la carpeta donde se creara el *htpassword* y también en las carpetas de administración y de programa del sistema. Una vez realizados los cambios se restablecen los permisos originales (755). Los permisos se otorgan mediante el comando *chmod*.
- Para los usuarios de *Windows* por lo general se cuenta con acceso total pero puede ser que en algunos casos interfiera el *firewall* o el antivirus por lo que se recomienda desactivarlos antes de realizar la operación.

Capítulo 7:

Conclusiones

"*Cuando miro a la sociedad, veo que la gente convierte las artes en ganancias comerciales; se considera a sí misma como mercancía e incluso lleva a cabo mejoras como si fuera un objeto de comercio. Distinguiendo lo superficial y lo sustancial, encuentro que esta actitud tiene menos realidad que la decoración.*"

Musashi Miyamoto – El libro de los cinco anillos

"*La luna que eclipsa la noche, el sol que encierra el cielo*"

Bleach – soul's guidebook

Capítulo 7

Conclusiones

En el presente capítulo se presentan las conclusiones finales de proyecto realizado. Como conclusiones generales, se hace énfasis en los limitaciones y alcances del sistema implementado. Como conclusiones particulares se presentan las dificultades y recomendaciones para la realización de un proyecto de tesis. Finalmente, se presentan las mejoras a las cuales el proyecto podría someterse para generar así un sistema más flexible y potente.

7.1 Conclusiones

Una vez concluida la propuesta realizada para la implementación del sistema semántico y debido a la experiencia surgida a través del desarrollo se puede llegar a dos tipos de conclusiones:

1. Conclusiones generales, en las cuales se presentan las ventajas y desventajas que surgen en la utilización del sistema semántico.
2. Conclusiones personales, las cuales surgen como resultado de la experiencia personal en la realización del proyecto de tesis, proporcionando algunas recomendaciones al momento de la realización de un proyecto similar.

7.1.1 Conclusiones generales

Desventajas

La principal desventaja al momento de implementar las clases es la comprensión de los sistemas orientados a objetos. Esto puede generar esquemas de relaciones innecesarias entre clases al momento de implementar un sistema, o relaciones demasiado complejas generando un sistema difícil de consultar.

Los permisos de usuarios deben ser mejor especificados, permitiendo o denegando el acceso los métodos de los lenguajes *DDL* y *DML*. Así como el permitir o denegar el acceso a diferentes bases de datos.

Los patrones de búsqueda de datos consumen mucha memoria al realizar la búsqueda de consulta en una clase si está compuesta por muchos objetos. Esto provoca que el sistema pierda rapidez y sea menos eficaz al momento de consulta de información.

Obviamente, el mayor problema comprende las cuestiones de seguridad dado que estas nunca se cumplen satisfactoriamente. El método propuesto mediante *htaccess* sólo reduce la posibilidad de riesgos, por lo que es necesario implementar mecanismos complementarios de seguridad.

Ventajas

Como se ha visto, el sistema resultante es una aplicación sencilla de utilizar, con la posibilidad de implementar relaciones más especificas entre clases, considerando como referencia las cinco relaciones propuestas originalmente [Antunes 1995] de las cuales sólo fueron implementadas cuatro.

La presentación de la información y diseño de la página es ajustable a las necesidades del usuario de manera rápida mediante los archivos *CSS*.

Se puede actualizar el sistema de acuerdo a las mejoras que se vayan presentando en los lenguajes base *Perl* y *MySQL*, los cuales además de ser lenguajes sumamente flexibles se encuentran en constante actualización.

En cuestión de costos de aplicación comparando con otros programas existentes como *Microsoft Office Excel* 2003, que permite establecer relaciones entre datos. Como producto único tiene un costo unitario usual de $3,870.00 [Excel 2007]. Para cuestiones de publicación de datos es necesario otras herramientas como *Front Page*, las cuales se encuentran incluidas en el paquete *Edición Office Edición Professional* 2003 que tiene un costo unitario usual de $8,518.00 [Office 2007].

Siendo el sistema, en costo de desarrollo sumamente económico debido al uso de software libre, en el cual la inversión en software fue nula.

7.2 Trabajos futuros

Se pueden plantear diversas modificaciones para mejorar el sistema genérico (SGUDEH). Algunos de ellos son sencillos y amplían los parámetros de operación del sistema. Algunas características que se podrían implementar son:

- Implementando ordenamiento de datos por columnas.
- Implementación de patrones de búsqueda de objetos por clase.
- La implementación de métodos de carga para imágenes y documentos, al mismo tiempo que se permita la visualización de imágenes y la descarga de archivos.
- Así como permitiendo patrones de seguridad aun mas amplios y no solamente limitar a usuarios del host local sino a usuarios de otros host específicos permitiendo la entrada a cualquier base de datos.
- Se pueden agregar otros métodos de relación de bases de datos propuestos.

Apéndices

"*Conócete a ti mismo*"
Sócrates

"*Andamos juntos, como gotas de agua, como cuerpos astrales. Nos rechazamos, como imanes, como los colores de la piel*"
Bleach Volumen 4

Apéndice A

Directivas de Apache 2.0

La tabla A.1 muestra algunas de las directivas más comunes para configurar Apache 2.0 [Informática 2006].

Tabla A.1 Principales directivas de Apache 2.0.

Directiva	Descripción
LoadModule	Carga el módulo correspondiente. Cada módulo habilita una serie de directivas.
ServerType	Tipo de ejecución del servidor, independiente o basada en *INETD*.
ServerName	Nombre completo del servidor.
Port	Puerto *TCP* gestionado por el servidor.
HostnameLookups	Búsqueda de clientes por nombre o por dirección *IP*. La búsqueda por nombres ralentiza la respuesta del *HTTPD*, es conveniente registrar los accesos por dirección *IP* y posteriormente revisarlos con programas estadísticos que soliciten los nombres al *DNS*.
User	Usuario ficticio propietario de los procesos del servidor.
Group	Grupo ficticio propietario de los procesos del servidor. Los usuarios reales que escriban páginas *Web* deben pertenecer a este grupo.
ServerAdmin	Dirección de correo del administrador del administrador del servicio.
ServerRoot	Directorio de configuración.
ErrorLog	Fichero histórico de errores (referido a *ServerRoot*).
CustomLog	Otros ficheros históricos (referido a *ServerRoot*).
KeepAlive.	Posibilidad de habilitar el uso de conexiones persistentes (recomendado)
StartServers	Número de procesos servidores que deben arrancarse (su función varía

	según el módulo usado para multiprocesos, *MPM*).Valor dependiente de la potencia del servidor.
MaxClients	Número máximo de procesos en ejecución o de clientes conectados (según el módulo MPM usado). También depende del tipo de servidor utilizado.
Listen	Permite la **escucha** en otros puertos para crear servidores virtuales.
<VirtualHost>	Directivas de configuración de un servidor virtual.
<Directory>	Directivas de configuración para accesos a directorios.
<Location>	Directivas de configuración de servicios asociados a *URLs*.
<Files>	Directivas de configuración asociadas a nombres de ficheros.
DocumentRoot	Directorio donde se encuentran los documentos principales del servidor.
UserDir	Directorio de los documentos personales de los usuarios.
DirectoryIndex	Archivo o programa que contiene el índice o la página principal de un directorio.
AddIcon	Incluye iconos que permiten identificar tipos de archivos.
Alias	Asocia nombres (alias) a directorios.
ScriptAlias	Indica los alias para directorios que incluyen programas CGI.
Redirect	Indica a los clientes que el documento está en una nueva URL externa. Nota: ver la documentación del módulo mod_rewrite ya que permite una mayor versatilidad para reescribir las URLs, tanto para redirecciones internas como externas.
ErrorDocument	Permite diseñar documentos que gestionan errores de acceso al servidor.

Asimismo, las distribuciones de Linux añaden algunas herramientas gráficas que ayudan a generar ficheros de configuración de forma más cómoda.

El servidor Apache incluye también varias directivas que permiten restringir o verificar el acceso a determinados documentos. Las subdirectivas incluidas en las cláusulas *<Directory>*, *<Files>* y *<Location>* y las equivalentes para expresiones regulares *<DirectoryMatch>*, *<FilesMatch>* y *<LocationMatch>*- establecen los permisos básicos

para acceder a directorios, ficheros y *URLs* específicos, respectivamente. La tabla muestra las directivas permitidas.

Tabla A.2 Directivas para configuración de acceso.

Subdirectiva	Descripción
Options	Opciones de acceso
All	Todas las opciones excepto *MultiViews* (valor por omisión).
ExecCGI	Se permite la ejecución de programas *CGI*.
FollowSymLinks	Permite seguir enlaces simbólicos (ignorado en <*Location*>).
Includes	Permite incluir ficheros y programas en documentos *HTML*.
IncludesNOEXEC	Permite incluir ficheros, pero no programas.
Indexes	Genera automáticamente un listado del directorio si en él no existe el fichero especificado en la directiva *DirectoryIndex*.
MultiViews	Soporta la negociación de contenidos especificada en el protocolo *HTTP*/1.1.
SymLinksIfOwnerMatch	Se siguen los enlaces simbólicos si el origen y el destino del enlace son del mismo usuario (ignorado en <*Location*>).
AllowOverride	Indica qué conjunto de opciones pueden solaparse mediante el fichero para la configuración de accesos al directorio (*.htaccess*).
AuthConfig	Configuración para autentificar usuarios con permiso de acceso.
<Limit>	Directivas aplicadas a determinados métodos de acceso del protocolo *HTTP*.
Order	Orden de preferencia para las cláusulas de permiso y denegación.
Allow	Permite el acceso a los nombres, direcciones IP o dominios indicados.
Deny	Deniega el acceso a los nombres, direcciones IP o dominios especificados.
Require	Indica qué usuarios o grupos tienen permiso de acceso.

La siguiente tabla muestra las directivas de configuración que gestionan los ficheros de usuarios y grupos en Apache 2.0.

Tabla A.3 Directivas para ficheros de usuarios.

Directiva	Descripción
AuthType	Declara cuál es el tipo de autentificación que va a utilizarse.
AuthName	Indica el nombre de autentificación enviado al programa cliente.
AuthUserFile	Establece el camino para el fichero de usuarios para la autentificación básica (módulo *mod_auth*).
AuthDigestFile	Establece el camino para el fichero de usuarios para la autentificación MD5 (módulo *mod_digest*). El administrador del servidor puede usar el programa htdigest (situado en el directorio *ServerRoot*/bin) para gestionar este tipo de ficheros. El algoritmo para codificación MD5 es más seguro que el usado por el método básico.
AuthGruopFile	Indica el camino para el fichero de grupos (módulo *mod_auth*).

Apéndice B

Permisos de MySQL

Los permisos de *MySQL* se dividen en tres grupos: datos, estructura y administración

Tabla B.1 Permisos de *Mysql*.

DATOS	
Select	Permiso para realizar consultas
Insert	Permiso para añadir datos.
Update	Permiso para editar y actualizar datos.
Delete	Permiso para borrar datos.
File	Permiso para que el usuario lea o escriba en ficheros residentes en el *Server*.
ESTRUCTURA	
Create	Permiso para crear nuevas tablas o bases de datos.
Drop	Permiso para borrar tablas o bases de datos.
Index	Permiso para indexar tablas (crear o borrar índices).
Alter	Permiso para cambiar la estructura de una tabla.
ADMINISTRACION	
Reload	Permiso para actualizar las tablas de permisos con *FLUSH*.
Shutdown	Permiso para apagar el servidor.
Process	Permiso para consultar los procesos del servidor con *MYSQLADMIN PROCESSLIST* ó *SHOW PROCESSLIST* (o cancelar esos procesos).
Grant	Permiso para que el usuario pueda conceder permisos a otros usuarios con *GRANT*.

Apéndice C

Archivos del SGUDEH

El sistema SGUDEH esta compuesto por distintos archivos, que trabajan de forma conjunta, se presenta de forma general las funciones realizadas por cada archivo.

C.1. Carpeta proyecto

Los siguientes 14 archivos se encuentran en la carpeta proyecto.

Archivo catalogo.cgi

Crea el formulario utilizado como catálogo de datos. Envía la información de los campos utilizando el método *POST* a la entrada estándar.

Archivo crea_tablas.cgi

Recibe los datos enviados por el archivo catalogo.cgi para crear los distintos tipos de clases. Está compuesto por seis funciones que realizan:

- es: implementa herencia de clases.
- tiene: implementa atribución.
- conjunto: implementa conjuntos.
- tupla: implementa tuplas.
- son_reservadas: regresa true ó false, compara si una variable es igual a las palabras reservadas.
- tiene_reservada: regresa true ó false, compara si un arreglo tiene un elemento igual a las palabras reservadas.

Archivo edicion.cgi

Este archivo es una biblioteca de funciones compuesta por 12 funciones que realizan las siguientes operaciones:

- es_grupo: regresa Y ó N, verifica en index_tablas si la tabla es un conjunto.

- seleccion: regresa los valores de una ID, consulta los valores de una tabla y se reenvía a la misma función en caso de tener relación con otra tabla.
- columnas_nombres: regresa los nombres de los atributos de una clase. Consulta el nombre de las columnas de una tabla, reenviándose a la misma función en caso de tener una relación con otra tabla.
- info_tupla: regresa el valor de la primera columna, que no sea ID, si es una relación con otra tabla se reenvía al método.
- es_tupla: regresa Y ó N, verifica se la tabla es una tupla.
- class_type: recibe el nombre de una tabla, regresa G en caso de ser un conjunto, T en caso de ser una tupla o R en cualquier otro caso.
- clases: regresa una lista de las tablas existentes en la base de datos, eliminando las tablas privadas.
- actualizar_index: agrega un nuevo nombre de una clase a un index.
- guardar_index: crea un registro de una clase en un index.
- info_index: regresa la información contenida en un index de una clase.
- existe_info: verifica si el nombre de una clase existe en el renglón de una clase guardada en un index.
- eliminar_index: elimina el nombre de una clase en los renglones de cada clase del index y elimina las columnas de relación en las tablas.

Archivo editar_definiciones.cgi

Crea un formulario para guardar, actualizar y eliminar las clases genéricas de la tabla my_definitions.

Archivo elimina.cgi

Crea los formularios para eliminar clases y atributos de clases. También realiza las operaciones de los formularios.

Archivo form_edicion.cgi

Crea el formulario para enviar los datos al archivo form_tablas.cgi por la entrada estándar, utilizando el método post.

Archivo form_tablas.cgi

Este archivo se crea los formularios para clases, conjuntos y tuplas, contiene las siguientes 10 funciones:

- crea_formulario: imprime un campo por cada columna de una tabla, si tiene una relación se reenvía al mismo método, utiliza la función imprime_formulario para los campos del formulario.
- imprime_formulario: dependiendo del tipo nativo del campo de la tabla imprime el campo, en caso de recibir información del campo, imprime el campo con la información por defecto.
- actualizar_formulario: extrae la información del campo e imprime un campo por cada columna de una tabla, si tiene una relación se reenvía al mismo método, utiliza la función imprime_formulario para los campos del formulario.
- opcion_grupo: cuando un campo es una referencia a una clase, conjunto o tupla se imprime en el formulario un campo de selección con las ID de la clase.
- form_tupla: inicia la impresión del formulario de una tupla, manda imprimir la información de las dos tablas que componen la tupla a la función datos_tupla.
- datos_tupla: imprime la información de las 4 primeras columnas si es una clase o tupla, si es un grupo se manda a la función imprime_grupo.
- imprime_grupo: imprime la información de las primeras 4columnas del conjunto. Extrae y pasa cada ID pertenecientes a la función imprime_grupoid.
- imprime_grupoid: imprime la información de las primeras 4 columnas por ID, con un campo de selección.
- form_grupo: inicia la impresión del formulario para agrupar información. Imprime el formulario si es tupla ó una clase. Si es un conjunto envia los datos a la funcion imprime_grupo_tupla.
- imprime_grupo_tupla: imprime la información de una clase o tupla, si es un grupo se manda a la función imprime_grupo_id.
- form_eliminar: inicio de los formularios para eliminación de datos de tablas. Imprime la información de la tabla si es tupla ó clase. Si es un conjunto envía los datos a la función imprime_grupo_tupla.

- imprime_grupo_id: imprime la información de cada ID junto con un campo de selección.

Archivo informacion.cgi

Se encarga de imprimir los datos de cada clase, así como un formulario dirigido hacia al archivo form_edicion.cgi. Contiene las siguientes 5 funciones:

- imprimir_tabla: imprime la información de una clase o tupla.
- es_grupo: regresa Y o N si la tabla es un conjunto
- imprime_grupo: imprime la información de un conjunto, extrae las ID y las envía a la función imprime_grupoid.
- imprime_grupoid: imprime la información por ID de un conjunto.
- Clases_relacionadas: busca en los index todas las clases que contienen el nombre de la clase.

Archivo llenado_tablas.cgi

Recibe la información de los formularios creados por form_tablas, contiene las siguientes 7 funciones:

- llena_formulario: crea una nueva ID por llamada, guarda la información mediante la función llena_tabla en cada columna, si es una relación a otra tabla, se reenvía a la función.
- llena_tabla: recibe nombre de campo, tabla, ID e información. Guarda la información en la columna correspondiente.
- crea_id: crea una ID en una tabla.
- actualiza_formulario: recibe una ID y guarda la información mediante la función llena_tabla en cada columna, si es una relación a otra tabla, se reenvía a la función.
- llenado_tupla: guarda cada ID en la columna correspondiente de la tabla de tipo tupla.
- guardar_grupo: guarda un conjunto de ID en la tabla de tipo conjunto, así como el nombre de grupo.
- eliminar_id: elimina una o varias ID de una tabla.

Archivo Reaver.pm

Este archivo es una biblioteca de funciones, cuenta con las 14 funciones siguientes:

- conexion: llama a la cookie y se conecta a la base de datos.
- extraeinfo: parte una cadena de datos, utilizando el carácter (&).
- existe_renglon: recibe el nombre de dos tablas, verifica si el nombre de la segunda tabla existe en la segunda columna de la primera tabla.
- existe_tabla: verifica si un nombre existe dentro de un arreglo
- es_tabla: regresa Y ó N, verifica si un nombre recibido es el nombre de una tabla.
- obtener_camponombres: regresa una cadena con el nombre de los campos de una tabla.
- existe_camponombre: regresa true ó false, verifica si el nombre recibido es el nombre de campo de una tabla dada.
- obtener_definicion: regresa la información de una clase genérica contenida en la tabla my_definitions.
- existe_dato: verifica si existe una patrón de datos dentro de una cadena
- lista_sinrepeticiones: elimina todos los componentes repetidos de un arreglo.
- elimina_caracter: elimina el carácter (`);
- imprime_tabla: imprime la información de una tabla.
- cambia_hexa: cambia los valores hexadecimales generados por el envío a través del servidor.
- *get_column_members*: función descargada de *codefetch* [codefetch 2006]. Obtiene los datos que puede adquirir un campo de tipo enum.

Archivo form_login.cgi

Este archivo sólo imprime un formulario para enviar los datos de contraseña y administrador mediante el método *POST* al archivo login.cgi.

Archivo login.cgi

Este archivo recibe los datos enviados por el archivo form_login.cgi, y crea una cookie con los datos del usuario como registro de inicio de sesión y registra el inicio de sesión en la tabla “usuarios” de la base de datos “profiles”. Si no existe la cookie y el registro de sesión,

no se puede trabajar con los archivos de la carpeta configuración. También permite la salida del administrador registrando la salida en la tabla “usuarios”.

Archivo pagina_inicio.cgi

Genera los menús de sección de cada página.

Archivo index.htm

Pagina HTML donde se incluye como iframe los archivos CGI de la carpeta administración.

Archivo mm_health_nutr.css

Archivo de hojas de estilo contiene los estilos de tablas, tipo de letras y la apariencia general de los archivos de la carpeta administración.

C.2. Carpeta administración

Los archivos siguientes se encuentran en la carpeta administración, dentro de la carpeta proyecto, los cuales permiten las funciones siguientes:

Archivo adm_databases.cgi

La estructura de este archivo permite generar distintos formularios a partir de un formulario inicial, que proporciona las opciones siguientes: Crear y eliminar una base datos, Seleccionar una de las bases de datos para que el sistema trabaje con ella.

Al crear una base de datos crea en ella las tablas siguientes: index_atrib, index_tablas y my_definitions. Este archivo cuenta con una función.

- crea_databases: crea o elimina una base de datos.

Archivo adm_users.cgi

La estructura de este archivo permite generar distintos formularios a partir de un formulario inicial, que proporciona las opciones siguientes: Crear usuarios, cambiar permisos de usuarios, actualizar contraseña de un usuario y eliminar usuarios.

Cuenta con 2 funciones

- exist_user: se encarga de verificar si el nombre de un usuario existe en la tabla users de la base de datos mysql.
- imprime_tabla: imprime la tabla users de la base de datos Mysql

Archivo configuración.cgi

Este archivo permite generar distintos formularios a partir de un formulario inicial. Los formularios permiten:

- Guardar la ruta hacia htpasswd y la ruta donde se guardaran los archivos de contraseñas.
- Generar los archivos de htaccess y generar los archivos de contraseña con htpasswd.
- Eliminar los archivos de htaccess y eliminar los archivos de contraseña.

Archivo form_login.cgi

Este archivo sólo imprime un formulario para enviar los datos de contraseña y administrador mediante el método *POST* al archivo login.cgi.

Archivo login.cgi

Este archivo recibe los datos enviados por el archivo form_login.cgi, y crea una cookie con los datos del administrador como registro de inicio de sesión y registrándola en la tabla "usuarios" de la base de datos "profiles". Si no existe la cookie y el registro de sesión, no se puede trabajar con los archivos de la carpeta configuración. También permite la salida del administrador registrando la salida en la tabla "usuarios".

La primera vez que se corre el archivo crea la base de datos "profiles", agregando en ella las tablas: "dbase_creadas", "usuarios" y "config_programa".

Archivo pagina_inicio.cgi

Genera los menús de sección de cada página.

Archivo index.htm

Pagina HTML donde se incluye como iframe los archivos CGI de la carpeta administración.

Archivo admin.css

Archivo de hojas de estilo contiene los estilos de tablas, tipo de letras y la apariencia general de los archivos de la carpeta administración.

Referencias

[Antunes 1995]

Alvaro A. Antunes, An Axiomatic Approach to Deductive Object Oriented Databases, Universidad de Heriot-Watt University Department of Computing and Electrical Engineering, 1995.

[Allen y Donald 2002]

R.Allen Wyke y Donald B. Thomas, Fundamentos de programación en Perl, Colombia: McGraw-Hill, 2002.

[Palacio 1999]

Juan Palacio, Perl, paginas Web interactivas, España: RA-MA, 1999.

[BUAP 2007]

BUAP. Notas de Programación de Sistemas: bases de datos. 17 de enero 2007. http://www.cs.buap.mx/~ygalicia/BD2007/BasesDatos.pdf

[GNU 2007]

El proyecto GNU. El sistema operativo GNU. 17 de enero 2007. http://www.gnu.org

[INFORMATICA 2000]

Informática. Seminario configuración de Apache 2. 30 de agosto de 2006. www.informatica.us.es/~ramon/articulos/SeminarioApache2.pdf

[INFORMATICO 2006]

El rinconcito informático. Bases de datos. 30 de agosto de 2006. http://www.elrinconcito.com/articulos/BaseDatos/BasesDatos.htm

[MySQL 2006]

MySQL. Manual de referencia 5.0. 25 de febrero de 2006. http://dev.mysql.com/doc/refman/5.0/es/index.html

[Office 2007]

Microsoft. Microsoft Office Excel 2003. 8 de abril de 2007. http://www.microsoft.com/spain/office/editions/howtobuy/professional.mspx

[Excel 2007]

Microsoft. Microsoft Office Edición Professional 2003. 8 de abril de 2007. http://www.microsoft.com/spain/office/products/excel/howtobuy/default.mspx

[DBAsupport 2007]

DBAsupport. Que es SQL. 17 de Enero de 2007. http://www.dbasupport.com.mx/index.php?option=com_content&task=view&id=74&Itemid=135

[PerldocCGI 2006]

Perldoc. CGI. 25 de Febrero de 2006. http://www.perldoc.perl.org/cgi.html

[PerldocDBI 2006]

Perldoc. DBI. 25 de Febrero de 2006. http://www.perldoc.perl.org/dbi.html

[Networksolutions 2007]

Network Solutions. Lenguaje SQL. 17 de Enero de 2007. http://networking-solution.blogspot.com/2006/12/lenguaje-sql.html

[Sromero 2006]

Sromero. Servidor Web con Apache. 25 de febrero de 2006. http://www.sromero.org/articulos/#lnxinet

[Informática 2006]

Informática. Configuración de Apache 2. 25 de Febrero de 2006. http://www.informatica.us.es/~ramon/articulos/SeminarioApache2.pdf

[codefetch 2006]

Codefetch. Codefetch source example. 13 de diciembre de 2006. http://perl.codefetch.com/search?qy=%23@+GET_COLUMN_INFO&lang=perl

[LAMP 2007]

Wikipedia. LAMP – Wikipedia, la enciclopedia libre. 13 de Marzo de 2007. http://es.wikipedia.org/wiki/LAMP

[Apache 2007]

Wikipedia. Servidor HTTP Apache – Wikipedia, la enciclopedia libre. 13 de Marzo de 2007. http://es.wikipedia.org/wiki/Servidor_HTTP_Apache

[Perl 2007]

Wikipedia. Perl – Wikipedia, la enciclopedia libre. 13 de Marzo de 2007. http://es.wikipedia.org/wiki/Perl