

# APRENDIENDO A CREAR WEBS CON PERL

por

Ubay Díaz Machín

Director de proyecto: Casiano Rodriguez de Leon

PRESENTADO CUMPLIENDO LOS REQUISITOS PARA  
PROYECTO FIN DE CARRERA

EN

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA  
UNIVERSIDAD DE LA LAGUNA  
LA LAGUNA, TENERIFE, ESPAÑA

30 JUNIO, 2008

© Copyright por Ubay Díaz Machín, 2008

Se concede permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre de GNU, Versión 1.2 o cualquier otra versión posterior publicada por la Free Software Foundation; sin Secciones Invariantes ni Textos de Cubierta Delantera ni Textos de Cubierta Trasera. El texto oficial de la licencia se encuentra en el apéndice **A**



UNIVERSIDAD DE LA LAGUNA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

El abajo firmante certifica que ha leído el “**Aprendiendo a crear webs con Perl**” realizado por **Ubay Díaz Machín**, y ya que cumple con los requisitos para un **Proyecto Fin de Carrera** recomienda a la Escuela Técnica Superior de Ingeniería Informática la aceptación del mismo.

Día: 30 Junio, 2008

Director de proyecto:

---

Casiano Rodriguez de Leon

Lectores:

---

Francisco Almeida Rodríguez

---

Elena Sanchez Nielsen

---

Coromoto León Hernández



*Dedicado a las familias y amigos de los autores.*



# Índice general

Índice de cuadros	XIII
Índice de figuras	XIX
Resumen	XXI
Agradecimientos	XXIII
<b>I Introducción</b>	<b>1</b>
<b>Capítulo 1. Introducción</b>	<b>3</b>
1.1. Que se piensa hacer . . . . .	3
1.2. Que se ha usado . . . . .	4
<b>II Introducción a SQL</b>	<b>5</b>
<b>Capítulo 2. ¿Qué Es?</b>	<b>7</b>
<b>Capítulo 3. Historia</b>	<b>9</b>
<b>Capítulo 4. Componentes</b>	<b>11</b>
4.1. Lenguaje de Definición de Datos (LDD o DDL) . . . . .	11
4.1.1. Crear . . . . .	11
4.1.2. Eliminar . . . . .	13
4.1.3. Actualizar . . . . .	13
4.1.4. Vaciar . . . . .	14
4.2. Lenguaje de Manipulación de Datos (LMD o DML) . . . . .	14
4.2.1. Inserción . . . . .	14
4.2.2. Actualizar . . . . .	15
4.2.3. Eliminación . . . . .	15
4.2.4. Selección . . . . .	16

<b>Capítulo 5. Ejemplos</b>	<b>19</b>
5.1. Creando Tablas . . . . .	19
5.2. Modificando Tablas . . . . .	20
5.3. Insertando Datos . . . . .	21
5.4. Eliminación de Datos . . . . .	21
5.5. Actualizando Datos . . . . .	22
<b>III Introducción a SQLite</b>	<b>23</b>
<b>Capítulo 6. Historia</b>	<b>25</b>
<b>Capítulo 7. ¿Quién Usa SQLite?</b>	<b>27</b>
7.1. Ejemplo de Proyectos donde es usado . . . . .	27
<b>Capítulo 8. Arquitectura de SQLite</b>	<b>29</b>
8.1. Interface . . . . .	29
8.2. El Compilador. Procesador de Consultas. . . . .	29
8.3. Máquina Virtual . . . . .	30
8.4. Interface de Almacenamiento . . . . .	30
<b>Capítulo 9. Ventajas y Defectos</b>	<b>31</b>
<b>IV DBI</b>	<b>33</b>
<b>Capítulo 10. ¿Qué es?</b>	<b>35</b>
<b>Capítulo 11. Funcionamiento</b>	<b>37</b>
11.1. Composición de DBD . . . . .	37
<b>Capítulo 12. Métodos para Manipular Datos en DBI</b>	<b>41</b>
12.1. Preparando la Consulta . . . . .	41
12.1.1. La Función prepare() . . . . .	41
12.2. Ejecutando . . . . .	43
12.2.1. execute() . . . . .	43
12.2.2. do() . . . . .	44

12.3. Mostrando . . . . .	44
12.3.1. Retornando un Array: fetchrow arrayref . . . . .	44
12.3.2. Seleccionar e Imprimir . . . . .	46
12.3.3. Seleccionando Columnas . . . . .	47
12.3.4. Seleccionando una Tabla Entera . . . . .	47
<b>Capítulo 13. DBD Actuales</b>	<b>49</b>
<b>Capítulo 14. Ejemplos de Códigos con Perl</b>	<b>51</b>
<b>V DBIx::Class</b>	<b>57</b>
<b>Capítulo 15. ¿Qué es?</b>	<b>59</b>
<b>Capítulo 16. Características</b>	<b>61</b>
16.1. Manipulación de Relaciones . . . . .	61
16.2. Definición de Estructura . . . . .	61
16.3. Búsqueda de Información en la Base de Datos . . . . .	61
16.4. Mostrar la Información . . . . .	61
<b>Capítulo 17. Componentes</b>	<b>63</b>
17.1. DBIx::Class::Schema . . . . .	64
17.2. DBIx::Class::Relationship . . . . .	64
17.2.1. Claves Ajenas: . . . . .	64
17.2.2. Relaciones 1 a muchos (has_many). . . . .	65
17.2.3. Relaciones 1 a 1. . . . .	66
17.2.4. Relaciones Muchos a Muchos. . . . .	67
17.3. DBIx::Class::ResultSet . . . . .	67
17.3.1. La familia de métodos search. . . . .	68
17.3.2. Iteradores . . . . .	68
17.3.3. Manipulando Datos . . . . .	69
<b>Capítulo 18. Ejemplo de Uso</b>	<b>71</b>

<b>VI</b>	<b>Template Toolkit</b>	<b>79</b>
	<b>Capítulo 19. ¿Qué Es?</b>	<b>81</b>
	<b>Capítulo 20. Cosas Básicas del Lenguaje</b>	<b>83</b>
	<b>Capítulo 21. Comentarios</b>	<b>85</b>
	<b>Capítulo 22. Directivas</b>	<b>87</b>
	22.1. Procesado de Ficheros . . . . .	87
	22.1.1. INSERT . . . . .	87
	22.1.2. INCLUDE . . . . .	88
	22.1.3. PROCESS . . . . .	89
	22.2. WRAPPER . . . . .	90
	22.3. BLOCK . . . . .	92
	22.4. Filtros y Plugins . . . . .	93
	22.4.1. FILTER . . . . .	93
	22.4.2. USE . . . . .	94
	<b>Capítulo 23. Variables</b>	<b>97</b>
	23.1. Escalares . . . . .	97
	23.2. Listas . . . . .	97
	23.3. Hashes . . . . .	98
	23.4. Métodos Virtuales . . . . .	99
	23.4.1. Métodos Escalares . . . . .	99
	23.4.2. Métodos referidos a Listas . . . . .	99
	23.4.3. Métodos referidos a Hash . . . . .	100
	<b>Capítulo 24. Control de Flujo</b>	<b>103</b>
	24.1. Sentencias Condicionales . . . . .	103
	24.1.1. IF, ELSIF, ELSE y UNLESS . . . . .	103
	24.1.2. SWITCH y CASE . . . . .	103
	24.2. Bucles . . . . .	103
	24.2.1. FOREACH . . . . .	104
	24.2.2. WHILE . . . . .	104

<b>Capítulo 25. Generando Páginas Dinámicas</b>	<b>107</b>
<b>VII Catalyst</b>	<b>109</b>
<b>Capítulo 26. Introducción</b>	<b>111</b>
26.1. Filosofía . . . . .	112
26.2. Servidores Webs soportados . . . . .	112
26.2.1. Pasando de Desarrollo a Producción . . . . .	113
26.3. Alternativas a Catalyst . . . . .	114
<b>Capítulo 27. Primeros Pasos en Catalyst</b>	<b>117</b>
27.1. Instalando Catalyst . . . . .	117
27.1.1. Instalándolo como un paquete Debian . . . . .	117
27.1.2. Instalando mediante CPAN . . . . .	117
27.2. Creando lo Básico de Catalyst . . . . .	118
27.2.1. Los Primeros Pasos . . . . .	119
27.3. Haciendo un Hola Mundo . . . . .	122
27.3.1. Añadiendo una Vista . . . . .	124
27.3.2. Añadiendo el controlador . . . . .	125
<b>Capítulo 28. Expandiendo el Ejemplo</b>	<b>131</b>
28.1. Añadiendo la Base de Datos . . . . .	131
28.1.1. Creando la Base de Datos . . . . .	131
28.1.2. Creando el Esquema de la Base de Datos en Catalyst . . . . .	133
28.2. Mostrando los Datos Almacenados . . . . .	140
28.2.1. Mostrando Item . . . . .	140
28.2.2. Mostrando Toda la Información . . . . .	143
28.3. Insertando Datos y Actualizando . . . . .	145
28.4. Eliminado una Entrada . . . . .	152
<b>Capítulo 29. Listado de Plugins</b>	<b>155</b>
<b>Capítulo 30. Notas Importantes</b>	<b>157</b>

<b>VIII Conclusiones</b>	<b>159</b>
<b>Capítulo 31. Conclusiones</b>	<b>161</b>
31.1. Lo Aprendido por el Autor . . . . .	161
31.2. Reflexiones sobre los utilizado . . . . .	161
<b>IX Bibliografía</b>	<b>163</b>
Bibliografía	165
<b>X Apéndices</b>	<b>167</b>
Apéndice A. GNU Free Documentation License	169

## Índice de cuadros

3.1.	Historial de Versiones de SQL[9] . . . . .	10
4.1.	Cláusulas para SELECT . . . . .	16
4.2.	Operadores de Comparación . . . . .	17
4.3.	Funciones de Agregado para SELECT . . . . .	17
5.1.	Creando la Tabla ITEM . . . . .	20
5.2.	Modificando la Tabla COLECCION . . . . .	20
5.3.	Insertando Datos . . . . .	21
5.4.	Eliminando Datos . . . . .	22
5.5.	Actualizando Datos . . . . .	22
9.1.	Comparativa de Ventajas y Desventajas de SQLite . . . . .	31
13.1.	Listado de DBD . . . . .	50
14.1.	Ejemplo de Uso de DBI en el entorno de SQLite . . . . .	52
14.2.	Ejecución del anterior Código de SQLite . . . . .	53
14.3.	Ejemplo de Uso de DBI en el entorno de MySQL[19] . . . . .	54
14.4.	Ejemplo de Uso de DBI en el entorno de Amazon[19] . . . . .	55
17.1.	Forma de Establecer una Relación de Clave Ajena . . . . .	65
17.2.	Forma de Establecer una Relación de 1 a muchos . . . . .	65
17.3.	Ejemplo de Establecer una Relación de 1 a muchos . . . . .	65
17.4.	Forma de Establecer una Relación de 1 a 1 con Borrado en Casacada . . . . .	66
17.5.	Ejemplo de Establecer una Relación de 1 a 1 con Borrado en Casacada . . . . .	66
17.6.	Forma de Establecer una Relación de 1 a 1 con existencia en las dos tablas . . . . .	66
17.7.	Ejemplo de Establecer una Relación de 1 a 1 con existencia en las dos tablas . . . . .	67
17.8.	Forma de Establecer una Relación de muchos a muchos . . . . .	67

17.9.	Ejemplo de Establecer una Relación de muchos a muchos . . .	67
18.1.	Esquema de Base de Datos en SQLite . . . . .	72
18.2.	Árbol de Directorios para representar la Base de Datos . . .	73
18.3.	El Fichero Main.pm . . . . .	73
18.4.	El fichero Places.pm . . . . .	73
18.5.	El Fichero Weather . . . . .	74
18.6.	El fichero Wiki.pm . . . . .	74
18.7.	El Fichero Enlaces.pm . . . . .	75
18.8.	El Fichero Basesdatos.pl . . . . .	76
18.9.	La continuación del Fichero Basesdatos.pl . . . . .	77
18.10.	La Continuación del Fichero Basesdatos.pl . . . . .	78
21.1.	Ejemplo de Comentario . . . . .	85
22.1.	Ejemplo de como indicar las Rutas de Búsqueda de Ficheros	87
22.2.	Ejemplo de uso de Insert en Template Toolkit . . . . .	88
22.3.	Ejemplo de como usar INCLUDE en Template Toolkit . . .	88
22.4.	Muestra del Comportamiento de las Variables en INCLUDE	89
22.5.	Ejemplo de como usar PROCESS en Template Toolkit . . .	89
22.6.	Muestra del Comportamiento de las Variables en PROCESS	90
22.7.	Muestra del Comportamiento de WRAPPER . . . . .	91
22.8.	Ejemplo de como usar PROCESS en Template Toolkit . . .	91
22.9.	Muestra del Comportamiento de las Variables en PROCESS	92
22.10.	Formas de Utilización de Block . . . . .	93
22.12.	Algunos Filtros interesantes incluidos en Template Toolkit.	94
22.11.	Formas de Utilización de Filter . . . . .	94
22.14.	Algunos Plugins interesantes incluidos en Template Toolkit.	95
22.13.	Forma de uso de USE . . . . .	95
23.1.	Uso de Variable Escalar en Template Toolkit . . . . .	97
23.2.	Uso de Listas en Template Toolkit . . . . .	98
23.3.	Uso de Listas en Template Toolkit . . . . .	98
24.1.	Uso de sentencias Condicionales IF Template Toolkit . . . .	103

24.2.	Uso de sentencias Condicionales SWITCH Template Toolkit	104
24.3.	Sintaxis de FOREACH en Template Toolkit . . . . .	104
24.4.	Posibilidades de iterator Template Toolkit . . . . .	105
24.5.	Sintaxis de WHILE en Template Toolkit . . . . .	106
25.1.	El Hola Mundo con Template Toolkit Versión 1 . . . . .	108
26.1.	Ejemplo de FastCGI . . . . .	113
26.2.	Ejemplo de FastCGI . . . . .	114
26.3.	Primer Cuadro de alternativas a Catalyst . . . . .	115
26.4.	Segundo Cuadro de alternativas a Catalyst . . . . .	116
27.1.	Instalando Paquete Debian [21] . . . . .	117
27.2.	Instalando Mediante CPAN . . . . .	118
27.3.	Creación de Esqueleto de Catalyst . . . . .	119
27.4.	Esqueleto Inicial de Catalyst . . . . .	121
27.5.	Ejecución Del Servidor Incluido en Catalyst, la primera Vez	122
27.6.	Petición de Web Inicial de Catalyst . . . . .	124
27.9.	Creando el Controlador de Hola Mundo [18] . . . . .	125
27.10.	Creacion de la Vista 1 <b>/lib/Controller/hola.pm</b> . . . . .	127
27.11.	Creación de la Vista 2 <b>/lib/Controller/hola.pm</b> . . . . .	128
27.7.	Creación de la Vista . . . . .	129
27.8.	Creando el Template de Hola Mundo <b>/root/src/hola.tt2</b> .	130
28.1.	Creación de la Base de Datos de Ejemplo . . . . .	132
28.2.	Inserción de datos Base de Datos de Ejemplo . . . . .	133
28.3.	Clases que utilizaran nuestra Base de Datos situado en el Fichero <b>/lib/EjemploDB.pm</b> . . . . .	134
28.4.	Clases que representa la Tabla Info situado en el Fichero <b>/lib/EjemploDB/Info.pm</b> . . . . .	135
28.5.	Clases que representa la Tabla Tipo situado en el Fichero <b>/lib/EjemploDB/Tipo.pm</b> . . . . .	136
28.6.	Clases que representa la Tabla Localización situado en el Fichero <b>/lib/EjemploDB/Localizacion.pm</b> . . . . .	136

28.7.	Clases que representa la Tabla Item situado en el Fichero <b>/lib/EjemploDB/Item.pm</b> . . . . .	137
28.8.	Clases que representa la Tabla Posee situado en el Fichero <b>/lib/EjemploDB/Posee.pm</b> . . . . .	138
28.9.	Creando el Modelo . . . . .	139
28.10.	Árbol de Directorios del Modelo . . . . .	139
28.11.	Los Nuevos Modelos añadidos . . . . .	140
28.12.	Añadiendo el Controlador Item . . . . .	141
28.13.	Función para el Listado de Contenido de Item situado en el Fichero <b>/lib/Controller/Item.pm</b> . . . . .	141
28.14.	Template Inicial para el Tratamiento de Item situado en el Fichero <b>/root/src/Item/list.tt2</b> . . . . .	142
28.15.	Relación Añadida a la Tabla Item situado en el Fichero <b>/lib/EjemploDB/Item.pm</b> . . . . .	143
28.16.	Template Modificado para Mostrar más Datos de Item si- tuado en el Fichero <b>/root/src/list.tt2</b> . . . . .	144
28.17.	Fichero de configuración del Formulario[23], situado en el Fichero <b>/root/forms/items/edit.fb</b> . . . . .	146
28.18.	Estableciendo campos de formulario de forma dinámica si- tuado en el Fichero <b>/lib/Controller/Item.pm</b> . . . . .	147
28.19.	Template que Procesa el Formulario situado en el Fichero <b>/usr/src/item/edit.tt2</b> . . . . .	147
28.20.	Template Variado en el Fichero <b>/root/src/Item/list.tt2</b> .	148
28.21.	Controlador Actualización Inserción de Información situa- do en el Fichero <b>/lib/Ejemplo/Controller/Item.pm</b> . .	149
28.22.	Controlador Actualización Inserción de Información Conti- nuación en el Fichero <b>/lib/Ejemplo/Controller/Item.pm</b>	150
28.23.	Controlador Actualización Inserción de Información parte Final en el Fichero <b>/lib/Ejemplo/Controller/Item.pm</b>	151
28.24.	Template Modificado para permitir eliminar Datos de Item en el Fichero <b>/root/src/item/list.tt2</b> . . . . .	152

28.25. Controlador para la Eliminación de Items situado en el Fi-  
chero **/lib/Ejemplo/Controller/Item.pm** . . . . . 153



## Índice de figuras

5.1.	Diagrama Entidad Relación de la Base de Datos de Ejemplo	19
6.1.	Cañón Vulcan Anti Misiles, sistema donde se vio la necesidad de algo como SQLite . . . . .	25
8.1.	Arquitectura de SQLite . . . . .	29
11.1.	Manipuladores DBI . . . . .	37
11.2.	Ejemplo de Diferentes Instancias, por cada conexión a diferentes Bases de Datos . . . . .	38
11.3.	Ejemplo de Diferentes Instancias, por cada Operación realizada. . . . .	39
18.1.	Diagrama de Base de Datos de Ejemplo . . . . .	71
26.1.	Logotipo del Framework Catalyst. . . . .	111
26.2.	Flujo de Trabajo de Catalyst . . . . .	112
27.1.	Esquema Resumen del Ejemplo para Catalyst . . . . .	118
27.2.	Web Por defecto Generada por Catalyst . . . . .	123
27.3.	Hola Mundo en Catalyst . . . . .	126
28.1.	Web inicial con el Listado de Item . . . . .	143
28.2.	Listado del Contenido de Item incluido información Adicional.	145
28.3.	Ejemplo de Formulario para la Manipulación de Items . . .	152
28.4.	Muestra de Borrado de Item . . . . .	153



## Resumen

Para crear aplicaciones web se tienen múltiples formas. Una de las estrategias más eficientes es el uso de un Framework. En el caso de este trabajo se ha optado por el Framework Catalyst que ha sido realizado mediante el lenguaje Perl y con la filosofía de este. En Catalyst se busca no tener que rehacer las cosas y reutilizar la mayor cantidad posible de código. También se intenta hacer las cosas de una forma diferente casi como si no se trabajara en formato Web sino de una forma más dirigida hacia la programación debido al uso de la arquitectura Modelo Vista Controlador (MVC). El MVC permite separar la forma de tratar la información de la lógica de negocio y de la forma de mostrar la información.



## Agradecimientos

Quiero agradecer a mis Padres por todos estos años de sacrificio para que yo pudiera asistir a la facultad y conseguir adquirir los conocimientos que me han permitido redactar este documento. Así mismo también les doy las gracias por la veces que con su apoyo consiguieron que en tiempos duros no perdiera la fe en mi y siguiera adelante.

También quiero dar las gracias a Casiano por aceptar a dirigirme en este trabajo y aguantar muchas de mis manías a la hora de hacer las cosas e impulsarme a obtener más conocimientos sobre la informática.

A los miembros del tribunal, Coro, Elena y Paco por ser miembros de este tribunal y en muchas ocasiones en el pasado aguantar todas mis preguntas.

Por último también quiero agradecer a todos mis amigos de la facultad que me han acompañado todos estos años y que espero que sigan siendo mis amigos por muchos años, en especial a Raúl que me proporcionó la plantilla para redactar este documento. Y muy en especial a Javi por ser un amigo fiel dispuesto a ayudar en todo momento sin querer nada a cambio.

Y no quiero olvidarme de dar las gracias a Dios y a mis abuelas que me han cuidado siempre.



# Parte I

## Introducción



# Capítulo 1

## Introducción

### 1.1. Que se piensa hacer

La idea de este trabajo ha sido la de ilustrar al lector en otra forma de utilizar Perl, las Bases de Datos y la creación de aplicaciones Webs. Para ello se intentará explicar de una forma descendente intentando que el lector pueda tener claro como poder crear su propia aplicación. La estructura que se ha utilizado es la que a continuación se explica. El lector debe poseer algún conocimiento en programación y haber tenido alguna experiencia con el lenguaje Perl.

- En el primer apartado se explican una serie de conceptos como pueden ser los tipos básicos de relaciones y el lenguaje SQL, para poder realizar las operaciones básicas de manipulación de la información de a base de datos.
- Para poder realizar el trabajo se necesitaba una plataforma de trabajo que incluya una base de datos. Se ha elegido SQLite ya que aunque no es el sistema más potente de mercado, si cumple lo necesario para mostrar el fin de este trabajo sin tener que ocuparnos de problemas de administración. Además al tener la base de datos en un fichero es sencillo el transporte de esta, dando la opción de poder trabajar en diferentes localizaciones.
- Para poder acceder a las bases de datos no sólo se tiene la opción de utilizar el lenguaje SQL, se debe disponer de herramientas que nos permiten manejarlas de una forma más sencilla. Por esa razón se introduce a DBI, un módulo Perl que permite manipular las bases de datos como si fueran objetos.
- Pero no sólo se tiene DBI, sino también se dispone de un módulo llamado DBIx::Class, que proporciona una interfaz orientada a la utilización sencilla de la Base de Datos.
- Tras conocer como manejar la Base de Datos nos ocupamos de como mostrar

la información. Para ello se utiliza otro modulo de Perl llamado Template::Toolkit que permite mostrar la información de una forma sencilla pero eficaz ya que tiene muchos métodos que nos permite mostrar de una forma clara y estructurada, con una sintaxis similar a la de HTML.

- Como punto final de este trabajo se trata el Framework Catalyst, que utilizando las herramientas anteriores permite crear aplicaciones web sin tener que preocuparse las partes más básicas de la creación web, ya que gracias a la existencias de múltiples Plugins la creación de formularios por ejemplo es muy simple así como las reglas de validación de estos formularios.

## 1.2. Que se ha usado

Para redactar este documento se han utilizado una serie de herramientas que se indican a continuación.

- Latex para redactar el documento[7].
- Kile para editar latex de forma correcta Latex[20].
- SVN sistema de control de versiones[8].
- KDESvn para manipular el SVN[6].
- Catalyst para aprender a sus interioridades[1].
- SQLite como sistema gestor de base de datos[5].

## Parte II

# Introducción a SQL



## Capítulo 2

### ¿Qué Es?

El Lenguaje de Consulta Estructurada (Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales, que nos permite realizar diferentes operaciones sobre estas. Una de sus funcionalidades es el manejo del álgebra y el cálculo relacional. Se debe tener en cuenta que al ser un lenguaje declarativo, no encontramos procedimientos. El orden de ejecución de las instrucciones internas es importante respecto a la eficiencia computacional. Aunque en teoría es un estándar, en la realidad cada motor de consulta, introduce sus diferencias, que pueden ir desde el orden en que se deben indicar las características de un atributo, hasta la forma de insertar datos.



## Capítulo 3

### Historia

Los orígenes del SQL están ligados a los de las bases de datos relacionales. En 1970 E. F. Codd propone el modelo relacional y asociado a este un sublenguaje de acceso a los datos basado en el cálculo de predicados. Basándose en estas ideas, los laboratorios de IBM definen el lenguaje SEQUEL (Structured English Query Language) que más tarde sería ampliamente implementado por el SGBD experimental System R, desarrollado en 1977 también por IBM. Sin embargo, fue Oracle quien lo introdujo por primera vez en 1979 en un programa comercial. El SEQUEL terminaría siendo el predecesor de SQL, siendo éste una versión evolucionada del primero. El SQL pasa a ser el lenguaje por excelencia de los diversos SGBD relacionales surgidos en los años siguientes y es por fin estandarizado en 1986 por el ANSI, dando lugar a la primera versión estándar de este lenguaje, el SQL-86 o SQL1. Al año siguiente este estándar es también adoptado por la ISO. Sin embargo este primer estándar no cubre todas las necesidades de los desarrolladores e incluye funcionalidades de definición de almacenamiento que se suprimen. En 1992 se lanza un nuevo estándar ampliado y revisado del SQL llamado SQL-92 o SQL2. En la actualidad el SQL es el estándar de facto de la inmensa mayoría de los SGBD comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general.[10], [9]

En la siguiente tabla 3.1<sup>1</sup>, se puede ver el historial de versiones de SQL, con sus principales cambios.

---

<sup>1</sup>Entrada en el Wikipedia Inglesa[10]

Año	Nombre	Alias	Comentarios
1986	SQL-86	SQL-87	Primera publicación hecha por ANSI. Confirmada por ISO en 1987.
1989	SQL-89		Revisión menor.
1992	SQL-92	SQL 2	Revisión mayor.
1999	SQL:1999	SQL 3	Se agregaron expresiones regulares, consultas recursivas (para relaciones jerárquicas), triggers y algunas características orientadas a objetos.
2003	SQL:2003		Introduce algunas características de XML, cambios en las funciones, estandarización del objeto sequence y de las columnas autonómicas.
2006	SQL:2006		ISO/IEC 9075-14:2006 Define las maneras en las cuales el SQL se puede utilizar conjuntamente con XML. Define maneras importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma XML. Además, proporciona facilidades que permiten a las aplicaciones integrar dentro de su código SQL el uso de XQuery, lenguaje de consulta XML publicado por el W3C (World Wide Web Consortium) para acceso concurrente a datos ordinarios SQL y documentos XML.

Cuadro 3.1: Historial de Versiones de SQL[9]

## Capítulo 4

### Componentes

El lenguaje SQL como cualquier otro necesita tener comandos , cláusulas , operadores y funciones. Ya sabemos que estas son las que nos vuelven locos. Por eso intentaré explicar más o menos como funcionan y para ello haré una diferenciación de los diferentes comandos. Por un lado podemos encontrar conjunto de componentes que se encargan de la definición de datos y su estructura, que se conoce como **Lenguaje de Definición de Datos** (LDD o DDL) y otro conjunto de componentes que se encargan de manipular toda la información, que se conoce como **Lenguaje de Manipulación de Datos** (LMD o DML). Explicaré los diferentes comandos agrupados según pertenezcan a LDD o LMD o a ambos. Comenzaré con LDD. Se han utilizado los siguiente recusos [3], [10] y [9]

#### 4.1. Lenguaje de Definición de Datos (LDD o DDL)

Este lenguaje se encarga de determinar la estructura del lenguaje, para lo que se realizan cuatro acciones básicas.

##### 4.1.1. Crear

Para crear estructuras se utiliza el comando CREATE, que no sólo crea Tablas, sino también objetos como vistas, índices y triggers. Para ello se usa la sintaxis siguiente:

#### CREATE TABLE

Descripción: Crea nuevas tablas

```
CREATE [ TEMPORARY | TEMP ] TABLE table (
  column type
  [ NULL | NOT NULL ] [ UNIQUE ] [ DEFAULT value ]
  [column_constraint_clause | PRIMARY KEY } [ ... ] ]
  [, ... ]
```

```
[, PRIMARY KEY ( column [, ...] ) ]

[, CHECK ( condition ) ]
[, table_constraint_clause ]
) [ INHERITS ( inherited_table [, ...] ) ]
```

Ejemplo de uso:

```
CREATE TABLE EJEMPLO ( ID VARCHAR2 PRIMARY KEY, ID1
  VARCHAR2 UNIQUE );
```

## CREATE TABLE AS

Descripción: Crea tablas, con los elementos insertados mediante un select

```
CREATE TABLE table [ (column [, ...] ) ]
  AS select_clause
```

## CREATE INDEX

Descripción: Crea un índice secundario, el cual nos permite un acceso más rápido a la información.

```
CREATE [ UNIQUE ] INDEX index_name ON table
  [ USING acc_name ] ( column [ ops_name ] [, ...] )
CREATE [ UNIQUE ] INDEX index_name ON table
  [ USING acc_name ] ( func_name( column [, ... ] ) [ ops_name ] )
```

## CREATE VIEW

Descripción: Crea una tabla virtual, a partir de los datos seleccionados por una sentencia SELECT. Permitiendo un manejo más sencillo de esa información.

```
CREATE VIEW view AS SELECT query
```

## CREATE TRIGGER

Descripción: Crea un Trigger, disparador, para que cuando se produzca un cierto suceso, se realice un operación indicada.

```
CREATE TRIGGER name { BEFORE | AFTER } { event [OR ...] }
  ON table FOR EACH { ROW | STATEMENT }
  EXECUTE PROCEDURE func ( arguments )
```

### 4.1.2. Eliminar

En este apartado intentaré definir algunos de los comandos disponible para eliminar objetos creados anteriormente.

## DROP TABLE

Descripción: Eliminación de la tabla definida anteriormente.

```
DROP TABLE name [, ...]
```

## DROP INDEX

Descripción: Elimina un índice creado anteriormente.

```
DROP INDEX index_name
```

## DROP VIEW

Descripción: Elimina una vista creada anteriormente.

```
DROP VIEW name
```

## DROP TRIGGER

Descripción: Elimina un trigger creado anteriormente.

```
DROP TRIGGER name ON table
```

### 4.1.3. Actualizar

Nos permite realizar cambios a los objetos que hemos creado anteriormente. A diferencia que en el caso de Eliminar, sólo podremos cambiar las tablas.

## ALTER TABLE

Descripción: Modifica una tabla anteriormente creada.

```
ALTER TABLE table [ * ]
    ADD [ COLUMN ] column type
ALTER TABLE table [ * ]
    ALTER [ COLUMN ] column { SET DEFAULT value | DROP DEFAULT }
ALTER TABLE table [ * ]
    RENAME [ COLUMN ] column TO newcolumn
ALTER TABLE table
    RENAME TO newtable
ALTER TABLE table
    ADD table constraint definition
```

### 4.1.4. Vaciar

Se refiere únicamente a vaciar una tabla, de todos los datos contenidos en ella, sin eliminar a esta.

## TRUNCATE

Descripción: Vaciar una Tabla.

```
TRUNCATE [ TABLE ] name
```

## 4.2. Lenguaje de Manipulación de Datos (LMD o DML)

En esta sección explicaré algunos de los comandos disponibles para manejar la información. Podremos subdividir en comandos de inserción , actualización, eliminación y selección, es decir, mostrar en forma de listado los registros.

### 4.2.1. Inserción

En estas líneas hablaré sobre como insertar datos en las tablas mediante sentencias SQL.

La forma básica de inserción es mediante el comando **INSERT**, que tiene varias formas de uso explicadas a continuación.

```
INSERT INTO tabla (columna1, [columna2, ... ])
VALUES (valor1, [valor2, ...])
```

Aquí se puede ver como la forma de inserción es la de indicar la tabla donde insertar, las columnas que serán rellenadas y los valores para estas columnas, pero teniendo en cuenta que los valores deben estar en el mismo orden que las columnas, es decir, el valor1 es para la columna 1.

Esta no es la única forma de inserción, porque también se puede ignorar las columnas, se indica la totalidad de valores de la tabla.

```
INSERT INTO tabla VALUES (valor1, [valor2, ...])
```

Así mismo también se pueden insertar varias filas de una tabla, si se separan los conjuntos de valores.

```
INSERT INTO tabla (columna1, [columna2, ... ])
VALUES (valor1a, [valor1b, ...]), (value2a, [value2b, ...]),
```

...

Cuando se necesita insertar una gran cantidad de datos, su origen puede ser una tabla ya existente usando el comando SELECT, que lista el contenido de tablas, para suministrar los valores.

#### 4.2.2. Actualizar

El comando usado es **UPDATE**, que es similar a INSERT, ya que básicamente también se debe indicar una tabla, unas columnas y unos valores para estas. De la forma siguiente.

```
UPDATE Tabla SET Campo1=Valor1, Campo2=Valor2,... CampoN=ValorN
WHERE Criterio;
```

Donde criterio debe ser una forma de localizar la fila o filas modificar. Por ejemplo nuestro DNI para poder cambiar el estado civil.

#### 4.2.3. Eliminación

El comando usado en este caso es **DELETE**, que es diferente al resto. Ya que sólo se indica la tabla en la cual se eliminaría un registro y la forma de indicarlo.

```
DELETE FROM class_name [WHERE qual];
```

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros.
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos.
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo.
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

Cuadro 4.1: Cláusulas para SELECT

#### 4.2.4. Selección

La selección es la obtención de información sobre la base de datos y cualquier operación sobre esta. Y el comando básico es **SELECT**, que tiene varias formas de usar, que van desde una consulta simple hasta el uso de consultas anidadas, etc. La forma básica sería:

```
SELECT [Columnas]
FROM [Tablas]
[WHERE condicion]
```

SELECT tiene una potencia extraordinaria. Me centraré en lo más útil: el agrupamiento, productos cartesianos y cláusulas para WHERE.

#### Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular. Véase la Tabla 4.1.

#### Operadores de Comparación

Véase la Tabla 4.2 .

Operador	Uso
<	Mayor que.
>	Menor que.
=	Igual que.
<>	Distinto que.
<=	Menor o igual que.
>=	Mayor o igual que.
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo.
IN	Utilizado para especificar registros de una base de datos.

Cuadro 4.2: Operadores de Comparación

Función	Descripción
AVG	Utilizada para calcular la media de valores numéricos.
COUNT	Cuenta el número de registros seleccionados.
MAX	Nos indica el valor máximo hallado.
MIN	Nos inidica el valor mínima hallado.
SUM	Calcula el sumatorio de los registros.

Cuadro 4.3: Funciones de Agregado para SELECT

### Funciones de Agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros. Vease la Tabla 4.3 .



## Capítulo 5

### Ejemplos

Para los ejemplos[5] me voy a basar en una Base de Datos para un sistema de catalogación de medios audio visuales, es decir, películas, música y demás. Siendo el diagrama Entidad Relación que se puede ver en la imagen 5.1

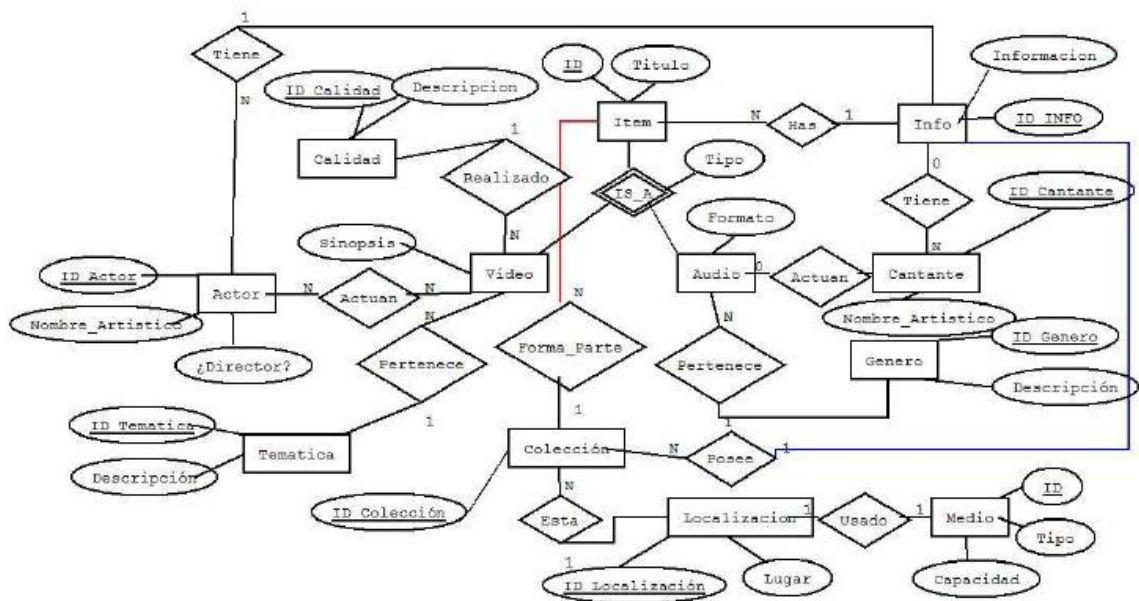


Figura 5.1: Diagrama Entidad Relación de la Base de Datos de Ejemplo

La idea es usar esta base de datos, para mostrar como usar los comandos anteriormente especificados. Pero no podemos crearla entera, porque no es el fin de este documento, al menos en este punto.

#### 5.1. Creando Tablas

Crearemos la tabla para los Item, la tabla Colección y luego la relación Forma Parte, mediante el uso de una tabla que tendrá como clave principal la clave de Item y secundaria la de Colección. Vease el Cuadro 5.1 .

```
SQLite version 3.4.2
Enter ".help" for instructions
sqlite> CREATE TABLE ITEM (
...> ID INTEGER PRIMARY KEY AUTOINCREMENT,
...> TITULO TEXT,
...> TIPO INTEGER);
sqlite> CREATE TABLE COLECCION (
...> ID_COLECCION INTEGER PRIMARY KEY AUTOINCREMENT);
sqlite> CREATE TABLE FORMA_PARTE (
...> ID_ITEM INTEGER PRIMARY KEY REFERENCES ITEM('ID')
...>                                     ON DELETE CASCADE,
...> ID_COLECC INTEGER REFERENCES COLECCION('ID_COLECCION')
...>                                     ON DELETE CASCADE);
```

Cuadro 5.1: Creando la Tabla ITEM

## 5.2. Modificando Tablas

Ahora voy a modificar la tabla Colección, para introducir una columna, que nos permita identificar la colección. Visto en el cuadro 5.2

```
sqlite> .schema COLECCION
CREATE TABLE COLECCION (
ID_COLECCION INTEGER PRIMARY KEY AUTOINCREMENT);
sqlite> ALTER TABLE COLECCION ADD COLUMN TITULO TEXT;
sqlite> .schema COLECCION
CREATE TABLE COLECCION (
ID_COLECCION INTEGER PRIMARY KEY AUTOINCREMENT, TITULO TEXT);
sqlite>
```

Cuadro 5.2: Modificando la Tabla COLECCION

### 5.3. Insertando Datos

Ahora insertaré información en las tablas Items, Colección y Forma Parte.  
Cuadro 5.3 .

```
sqlite>
sqlite> SELECT * FROM COLECCION;
sqlite> SELECT * FROM ITEM;
sqlite> SELECT * FROM FORMA_PARTE;
sqlite> INSERT INTO ITEM VALUES(null,'Uno',1);
sqlite> INSERT INTO ITEM VALUES(null,'Dos',2);
sqlite> INSERT INTO ITEM VALUES(null,'Tres',3);
sqlite> INSERT INTO COLECCION VALUES(null,'Uno');
sqlite> INSERT INTO COLECCION VALUES(null,'Dos');
sqlite> INSERT INTO FORMA_PARTE VALUES(1,1);
sqlite> INSERT INTO FORMA_PARTE VALUES(2,2);
sqlite> SELECT * FROM ITEM;
1|Uno|1
2|Dos|2
3|Tres|3
sqlite> SELECT * FROM COLECCION;
1|Uno
2|Dos
sqlite> SELECT * FROM FORMA_PARTE;
1|1
2|2
sqlite>
```

Cuadro 5.3: Insertando Datos

### 5.4. Eliminación de Datos

Ahora eliminaré registros de la Tabla Item. Cuadro 5.4

```
sqlite> SELECT * FROM ITEM;
1|Uno|1
2|Dos|2
3|Tres|3
sqlite> DELETE FROM ITEM WHERE ID=2;
sqlite> SELECT * FROM ITEM;
1|Uno|1
3|Tres|3
sqlite>
```

Cuadro 5.4: Eliminando Datos

### 5.5. Actualizando Datos

Se actualizará la información en la tabla Item. Cuadro 5.5

```
sqlite> SELECT * FROM ITEM;
1|Uno|1
3|Tres|3
sqlite> UPDATE ITEM SET TITULO='CUATRO' WHERE ID=3;
sqlite> SELECT * FROM ITEM;
1|Uno|1
3|CUATRO|3
sqlite>
```

Cuadro 5.5: Actualizando Datos

## Parte III

# Introducción a SQLite



## Capítulo 6

### Historia

Esta genial Base de Datos surgió en el campo de batalla, pero no en el mundo de la informática, sino en el real ya que el autor, Richard Hipp, estaba trabajando para General Dynamics, se encontraba realizando un proyecto para la U.S. Navy que había realizado un pedido de un sistema de guiado para los cañones anti misiles de sus barcos. Durante este trabajo vio que el sistema era muy pesado, ya que se usaba una Hewlett-Packard Unix y una base de datos Informix. Implicaba mucho tiempo tanto de configuración como instalación. Debido a esto Richard Hipp junto a un colega en Enero del 2000, comenzaron a hablar sobre la forma de realizar una base de datos empotrada, que fuera simple, pequeña, fácil de gestionar, pero que a su vez cumpliera en todo lo posible el Estándar SQL-92. En Agosto de ese año ya estaba disponible la versión 1 de SQLite, esta estaba basada en la librería GNU DBM B-Tree, que se encargaba de manejar el almacenamiento. En 2004 salió la versión 2 y en 2005 la versión 3.[\[22\]](#)

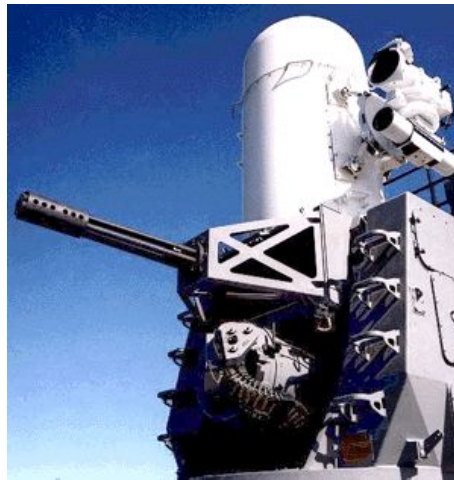


Figura 6.1: Cañón Vulcan Anti Misiles, sistema donde se vio la necesidad de algo como SQLite



## Capítulo 7

### ¿Quién Usa SQLite?

Aunque el creador no lo esperaba, vio con sorpresa y alegría que su trabajo era cada vez más utilizado en diferentes proyectos, como se puede ver en la lista siguiente.[\[22\]](#)

#### 7.1. Ejemplo de Proyectos donde es usado

- SQLite es usado por el entorno de base de datos Kexi como un motor de base de datos interno por defecto.
- SQLite se ha usado para guardar el índice para un set de DVD conteniendo todos los números publicados de la revista The New Yorker.
- Yum, la herramienta de gestión de paquetes de Fedora Core, ha cambiado a SQLite y pysqlite para el almacenamiento de datos y análisis de XML desde el núcleo de Fedora 4. De acuerdo con los comentarios de los usuarios, el incremento de funcionamiento es impresionante, así como la reducción en consumo de memoria.
- Los desarrolladores de OpenOffice.org han considerado incrustar SQLite en el modelo de base de datos de Base, pero esto depende en gran manera del progreso de sqlite-sdbc-driver, que está todavía en estado alpha. Actualmente han decidido [\[3\]](#) usar HSQLDB, pero la opción está todavía abierta siempre y cuando el driver sqlite-sdbc acabe siendo maduro y de confianza.
- Open Outlook Connector, un conector open source de Microsoft Outlook/MAPI ha sido hecho usando SQLite.
- Varias aplicaciones de Apple utilizan SQLite, incluyendo Apple Mail y el gestor de RSS que se distribuye con OS X. El software Aperture de Apple guarda la información de las imágenes en una base de datos SQLite, utilizando la API Core Data.

- Monotone, un sistema de control de versiones de libre distribución lo usa.
- El nuevo sistema de almacenamiento implementado en el núcleo de Mozilla (MozStorage) utiliza SQLite. Una versión futura de Mozilla Firefox hará uso de este sistema basado en SQLite para almacenar bookmarks y el histórico de navegación. Se está considerando también utilizar SQLite en Mozilla Calendar y Mozilla Thunderbird.
- PyKota, una solución gratuita de cuotas de impresión y contabilidad, puede utilizar SQLite como su base de datos.
- Amarok usa SQLite para guardar su colección de datos por defecto; en nuevas versiones, puede usarse bases de datos externas.
- XMMS2 usa SQLite para su Librería de Medios. La librería de medios consta de un índice de metadatos de las canciones en la librería, así como un lugar en el cuál almacenar listas de reproducción.
- SQLFilter, un plugin para OmniPeek, usa SQLite para indexar paquetes en una base de datos para poder ser consultada por medio de SQL.
- HaXe usa SQLite como parte de su servidor incluido.

## Capítulo 8

### Arquitectura de SQLite

La arquitectura de SQLite es sencilla pero robusta. Está compuesta por 8 módulos separados entre si, que se pueden agrupar en la interfase, el procesador de las consultas, una maquina virtual y la interfase de almacenamiento y con el Sistema Operativo[22], Figura 8.1.

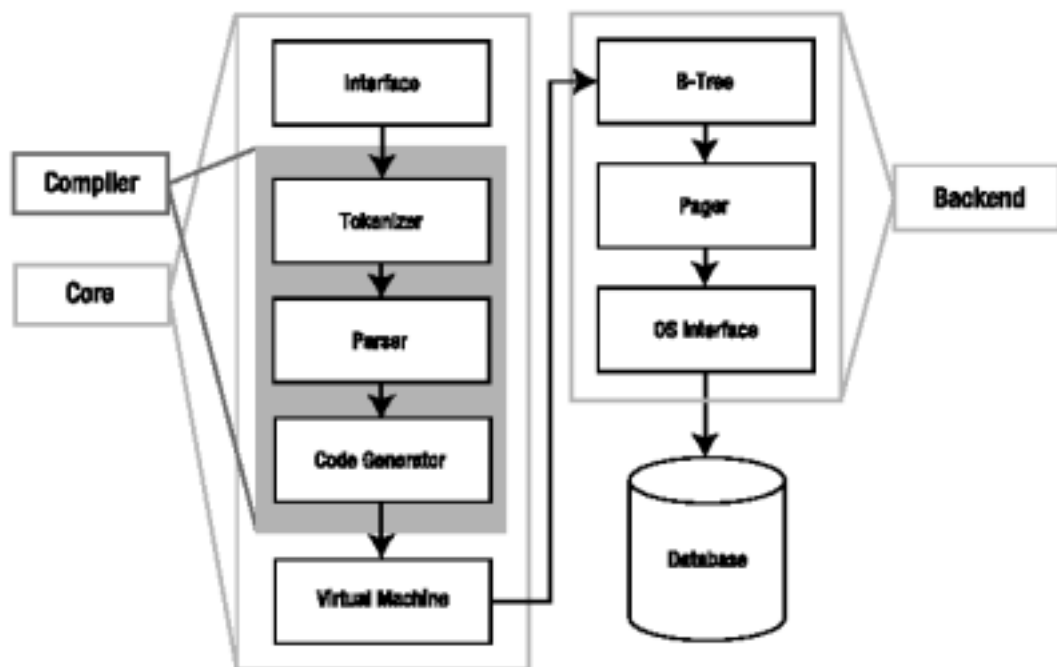


Figura 8.1: Arquitectura de SQLite

#### 8.1. Interface

Es una API en C, que nos permite manejar la base de datos.

#### 8.2. El Compilador. Procesador de Consultas.

Este módulo se encarga de comprobar la sintaxis de la sentencia SQL que le es pasada, y transformarla en una estructura comprensible por el sistema.

Es además responsable de planificar la forma en que se ejecuta para mejorar el rendimiento.[22]

### **8.3. Máquina Virtual**

Esta máquina está diseñada para el procesado de datos. Está compuesta por 128 Opcodes con los que se pueden formar cualquier consulta que queramos hacer.[22]

### **8.4. Interface de Almacenamiento**

Esta parte está implementado con un árbol B. [22]

## Capítulo 9

### Ventajas y Defectos

Como se puede ver en la tabla 9.1 [22]

<b>Ventajas</b>	<b>Inconvenientes</b>
Sin ningún tipo de configuración	No tiene control de de claves ajenas, pero se esperaba tener en el 2006.
Es muy portable encontrándose en muchas plataformas.	No se tiene un completo soporte de Triggers.
Es muy compacta el sistema de gestión, unos 250 KBytes	Sólo permite añadir nuevas columnas a una tabla, no eliminar o cambiar.
Es muy simple permitiendo incluirla en programas como una librería más.	Sólo tiene implementado los Left Join y no los Right.
Su código es de acceso público.	Las vistas no son actualizables.
Tiene un buen control de la concurrencia, orientado hacia lectura múltiple y escritura única.	El control de permisos es cosa del sistema de ficheros.
Puede escalar hasta los 2 Terabytes,	Su control de transacciones no es muy bueno.
Puede ser compartida mediante los sistemas de ficheros en red.	

Cuadro 9.1: Comparativa de Ventajas y Desventajas de SQLite



# Parte IV

## DBI



## Capítulo 10

### ¿Qué es?

Es un modulo de Perl, que posibilita a este lenguaje el acceso uniforme a diferentes bases de datos. Con el único fin que al escribir una aplicación que use bases de datos, no importe que esta sea Oracle, SQLite, MySQL,etc. Para ello nos provee de métodos con lo cuales realizar las operaciones más habituales.



## Capítulo 11

### Funcionamiento

El funcionamiento del sistema está basado en dos niveles. Teniendo un nivel que corresponde a la interfase de programación y otro que maneja toda las manipulaciones con las diferentes bases de datos soportadas, conocidos como DBD[19].

#### 11.1. Composición de DBD

Un DBD, es un objeto que nos permite la manipulación de una base de datos específica, con esto se quiere decir que esta es la parte que cambia según la base de datos usada. Así por ejemplo tendremos uno para MySQL otro para SQLite, etc. Este objeto tiene una estructura más compleja de lo que se puede imaginar, ya que está formado por tres capas. La primera es el **Driver Handle**, la segunda es **Database Handle** y la tercera es el **Statement Handle**.

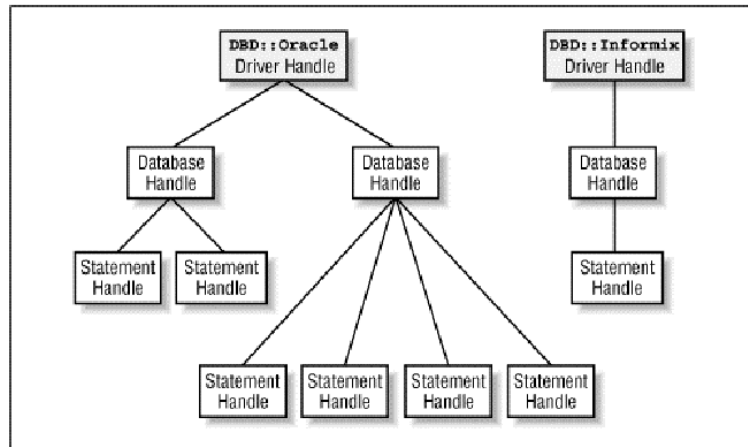


Figura 11.1: Manipuladores DBI

#### Driver Handle

Esto representa el driver cuando es cargado. En esta capa se deben encontrar estos métodos[19]. Que son,

`data_sources`

y

```
connect
```

.

```
dbi->data_sources();
```

```
dbi->connect();
```

## Database Handle

Básicamente este objeto aunque se encuentra englobado, por el método anterior 11.1, es en cierta forma independiente. Ya que se creará una instancia de este por cada conexión que creamos a una base de datos en concreto. Quiero decir, que aunque nuestro sistema gestor de base de datos sea, SQLite, en ellas pueden haber múltiples bases de datos, pues básicamente tendremos una instancia por cada uno de estas bases de datos y conexiones a ellas.

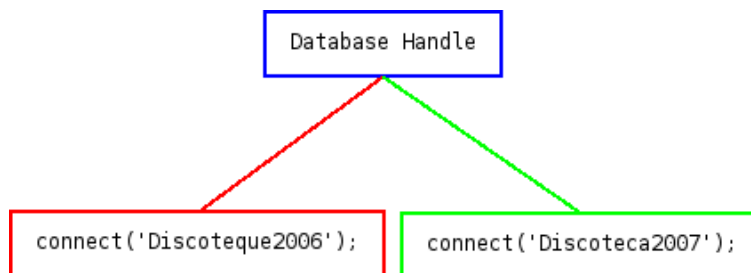


Figura 11.2: Ejemplo de Diferentes Instancias, por cada conexión a diferentes Bases de Datos

## Statement Handle

Este objeto como el anterior 11.1, este también está englobado en él. Ya que este se encarga de realizar cada una de las posibles operaciones a realizar sobre estas bases de datos. Por cada selección, o inserción, tendremos una instancia de este objeto.

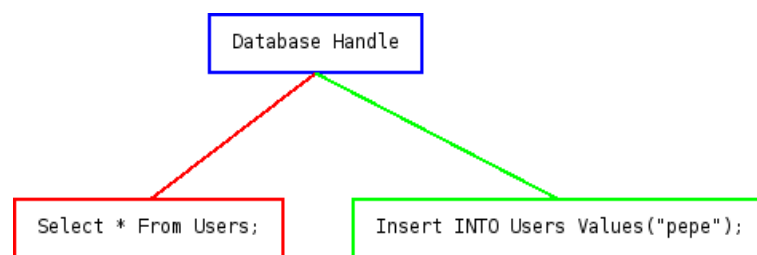


Figura 11.3: Ejemplo de Diferentes Instancias, por cada Operación realizada.



## Capítulo 12

### Métodos para Manipular Datos en DBI

Como luego veremos en el capítulo de Ejemplos de Códigos 14, con la misma filosofía de Perl, se tienen varias formas de realizar las cosas en DBI. Y en las siguientes líneas se verán algunas de ellas:

#### 12.1. Preparando la Consulta

En esta sección intentaré explicar lo mejor que pueda las formas de preparar las consultas. Pero primero surge la pregunta: ¿Qué es preparar las Consulta?, pues básicamente es asegurarnos que la consulta ha realizar, es posible y no tiene ningún problema que pueda surgir en la ejecución.

##### 12.1.1. La Función prepare()

Esta función básicamente comprueba que la consulta ha realizar este en un formato correcto. Para hacer esto tiene dos formas, una es cuando la consulta esta predefinida con anterioridad y la otra cuando la consulta se crea de forma dinámica.

#### Forma Predefinida

La forma predefinida es sencilla, ya que usando el objeto que se crea al conectar a la base de datos, crearemos otro que será la consulta “preparada”, como se puede ver a continuación.

```
#!/usr/bin/perl -w
#
# ch05/prepare/ex1: Simply creates a database handle and a statement handle
use DBI;
### The database handle
my $dbh = DBI->connect( "dbi:Oracle:archaeo", "username", "password" );
### The statement handle
```

```
my $sth = $dbh->prepare( "SELECT id, name FROM megaliths" );

exit;
```

## Forma Dinámica

Esta forma es la más utilizada, ya que nos permite realizar consulta donde los parámetros pueden ser variables que cambiarán según el momento. Como por ejemplo una consulta que nos muestre los usuarios nacidos en 1980 y que luego reutilizaremos para buscar a los nacidos en 1990. Para realizar esto, uno podría tener la tentación de incluir las variables en la cadena pasada a **prepare()**, pero esto no sería suficiente, ya que esto producirá error. Para poder realizar esto antes debemos tratar la información, para que tenga un formato adecuado, escapando cualquier marca o carácter extraño. Y la forma más usada es, con el objeto creado al conectar a la base de datos , pasarle el dato a tratar, como se puede ver en el ejemplo a continuación.

```
### This variable is populated from the online form, somehow...
my $siteNameToQuery = $CGI->param( "SITE_NAME" );
### Take care to correctly quote it for use in an SQL statement
my $siteNameToQuery_quoted = $dbh->quote( $siteNameToQuery );
### Now interpolate the variable into the double-quoted SQL statement
$sth = $dbh->prepare( "
    SELECT meg.name, st.site_type, meg.location, meg.mapref
    FROM megaliths meg, site_types st
    WHERE name = $siteNameToQuery_quoted
    AND meg.site_type_id = st.id
" );
$sth->execute( );
@row = $sth->fetchrow_array( );
...
```

Aunque existe otra forma de poder pasarle parámetros a nuestra consulta utilizando **bind\_param**, que realiza el quote al mismo tiempo que el parámetro es enviado a la base de datos, tras el envío de la consulta. Para esto nosotros sólo tenemos que marcar en la consulta los parámetros con `?`, como se puede ver en el ejemplo a continuación.

```
$sth = $dbh->prepare( "
    SELECT name, location
    FROM megaliths
    WHERE name = ?
    AND mapref = ?
    AND type LIKE ?
" );
$sth->bind_param( 1, "Avebury" );
$sth->bind_param( 2, $mapreference );
$sth->bind_param( 3, "%Stone Circle%" );
```

## 12.2. Ejecutando

En las siguientes líneas quiero expresar las diferentes formas de ejecutar consultas, pero teniendo en cuenta que como todo en la vida según el momento se deberá utilizar un método u otro. Así cuando estemos ejecutando una consulta del tipo Selección será mejor utilizar **execute()**, mientras que cuando ejecutemos una consulta de Inserción, Creación, Borrado o para resumir, cualquier acción que no devuelva datos será mejor utilizar **do()**.

### 12.2.1. execute()

Es la forma más sencilla, ya que tras utilizar `prepare()` se usa el objeto generado por este, se llamará al función `execute()`, para que ejecute la consulta, como se puede ver a continuación:

```
### Now interpolate the variable into the double-quoted SQL statement
$sth = $dbh->prepare( "
    SELECT meg.name, st.site_type, meg.location, meg.mapref
```

```

        FROM megaliths meg, site_types st
        WHERE name = $siteNameToQuery_quoted
        AND meg.site_type_id = st.id
    " );
    $sth->execute( );

```

### 12.2.2. do()

Esta función a diferencia de `execute`, se suele usar para las consultas que no produzcan salidas, como pueden ser las operaciones de inserción, borrado, actualización y demás. La forma de usar es pasar una cadena con la consulta, en la cual podremos pasar parámetros de forma indirecta. Como se puede ver en la siguiente línea.

```

my $rows_deleted = $dbh->do(q{
    DELETE FROM table
    WHERE status = ?
}, undef, 'DONE') || die $dbh->errstr;

```

## 12.3. Mostrando

En esta sección mostraré algunos de los métodos para poder mostrar los datos contenidos en nuestros sistemas de bases de datos.

### 12.3.1. Retornando un Array: `fetchrow_arrayref`

Este método básicamente nos devuelve los datos como si fueran un array. Dándonos la opción de ir obteniendo cada fila del array o todo el array. Como se puede ver en el siguiente código, se ilustra la obtención de los datos fila por fila.

```

### Fetch the rows of result data from the database
### as an array ref....
while ( $array_ref = $sth->fetchrow_arrayref ) {
    ### Print out a wee message....
    print "Megalithic site $array_ref->[0] is a $array_ref->[1]\n";
}
die "Fetch failed due to $DBI::errstr" if $DBI::err;

```

Y en el siguiente código se muestra el ejemplo de obtener todo un array.

```

### The stash for rows...
my @stash;
### Fetch the row references and stash 'em!
while ( $array_ref = $sth->fetchrow_arrayref ) {
    push @stash, $array_ref;      # XXX WRONG!
}
### Dump the stash contents!
foreach $array_ref ( @stash ) {
    print "Row: @$array_ref\n";
}

```

Otra forma de volcar la información mediante arrays, es la utilización de referencia y también se ha implementado un método adecuado. Como podemos ver en el código a continuación.

```

### The stash for rows...
my @stash;
### Fetch the row references and stash 'em!
while ( $array_ref = $sth->fetchrow_arrayref ) {
    push @stash, $array_ref;
}
### Dump the stash contents!
foreach $array_ref ( @stash ) {
    print "Row: @$array_ref\n";
}

```

Otro método interesante es el que nos vuelca la información en un formato de Hash de Perl, es decir, la estructura volcada sería un array donde las claves son los nombres de las columnas de las tablas seleccionadas. Como podemos ver en el código de ejemplo a continuación.

```

### Fetch rows into a hash reference
while ( $hash_ref = $sth->fetchrow_hashref ) {
    print "Megalithic site $hash_ref->{name} is a $hash_ref->{site_type}\n";
}

```

### 12.3.2. Seleccionar e Imprimir

Esta sección versa de un método, más concretamente sobre `dump_results()`, que aparte de seleccionar la información a mostrar, la imprime directamente. Con la opción de poder formatear la salida, pudiéndose indicar cosas como lo siguiente.

1. Tamaño máximo del Campo: Ejemplo = `¿35`
2. Separador de Línea: Ejemplo = `¿  
n`
3. Separador de Campos: Ejemplo = `¿:'`
4. Salida de los datos: Ejemplo = `¿'STDOUT'`

Pero estos parámetros son opcionales, pero no tienen por que ser usados. Como podemos ver en los siguiente códigos. Esto es el ejemplo sin usar formatos.

```
$sth = $dbh->prepare( "
    SELECT name, mapref, location
    FROM megaliths
    " );
$sth->execute( );
$rows = $sth->dump_results( );
```

Usando especificación de formatos.

```
### Prepare and execute the query
$sth = $dbh->prepare( "
    SELECT name, location, mapref
    FROM megaliths
    " );
$sth->execute( );
### Open the output file
open FILE, ">results.lis" or die "Can't open results.lis: $!";
### Dump the formatted results to the file
$rows = $sth->dump_results( 80, '\n', ':', \*FILE );
### Close the output file
close FILE or die "Error closing result file: $!\n";
```

### 12.3.3. Seleccionando Columnas

También existen forma se seleccionar una fila en únicamente. Este nos permite ahorrarnos el paso de preparar la consulta porque realiza toda la operación en un sólo paso, devolviendo un array con las columnas seleccionadas. Como podemos ver en el siguiente código de ejemplo.

```
### Assuming a valid $dbh exists...
( $name, $mapref ) =
    $dbh->selectrow_array( "SELECT name, mapref
                          FROM megaliths" );
print "Megalith $name is located at $mapref\n";
```

### 12.3.4. Seleccionando una Tabla Entera

Esta sección introduce a la forma de obtener de una consulta todos los elementos de una sola vez. Devolviendo una referencia que puede ser tratado como un array. Permittendonos incluso poder especificar las columnas que se quieren imprimir como explicaré más adelante.

El ejemplo siguiente es el uso de **fetchall\_arrayref** para obtener los datos.

```
#!/usr/bin/perl -w
#
# ch05/fetchall_arrayref/ex1: Complete example that connects to a database,
#                               executes a SQL statement, then fetches all the
#                               data rows out into a data structure. This
#                               structure is then traversed and printed.
use DBI;
### The database handle
my $dbh = DBI->connect( "dbi:Oracle:archaeo", "username", "password" , {
    RaiseError => 1
});
### The statement handle
my $sth = $dbh->prepare( " SELECT name, location, mapref FROM megaliths " );
### Execute the statement
$sth->execute( );
### Fetch all the data into a Perl data structure
```

```

my $array_ref = $sth->fetchall_arrayref( );
### Traverse the data structure and dump each piece of data out
###
### For each row in the returned array reference ...
foreach my $row (@$array_ref) {
    ### Split the row up and print each field ...
    my ( $name, $type, $location ) = @$row;
    print "\tMegalithic site $name, found in $location, is a $type\n";
}
exit;

```

Esta también nos permite usar parámetros para poder seleccionar las columnas a mostrar de la consulta realizada. Así por ejemplo si habían cuatro columnas en la consulta. Se puede seleccionar la columna 0 y 2, pasándole un array con estos números, de esta forma [0,2]. Y el siguiente código se muestra un ejemplo.

```

### The statement handle
my $sth = $dbh->prepare( " SELECT name, location, mapref FROM megaliths " );
### Execute the statement
$sth->execute( );
### Retrieve the name and location fields...
$array_ref = $sth->fetchall_arrayref( [ 0, 2 ] );

```

## Capítulo 13

### DBD Actuales

En esta sección mostraré un listado con algunos de los DBD, que podemos encontrar ahora mismo en CPAN. Que se encuentra en la tabla 13.1 .

<b>DBD</b>	<b>Descripción</b>
DBD::mSQL	Driver para el gestor MySQL
DBD::RAM	Permite volcar estructuras Perl, a diferentes tipos de ficheros como si fueran bases de datos.
DBD::ODBC	Manipulación del sistema ODBC de Microsoft.
DBD::Informix	Para el sistema Informix de IBM.
DBD::ADO	Un driver para Microsoft ADO (Active Data Objects).
DBD::LDAP	Driver para LDAP.
DBD::Oracle	Permite la manipulación de gestores Oracle..
DBD::Amazon	Permite acceder a la API de Amazon como si fuera una Base de Datos.
DBD::SQLite	Driver DBD para SQLite.
DBD::Pg	Driver DBD para PostgreSQL.

Cuadro 13.1: Listado de DBD

## Capítulo 14

### Ejemplos de Códigos con Perl

En esta sección se mostrarán ejemplos de uso del DBI, con diferentes gestores de Bases de Datos, comenzando con SQLite del cual tenemos un código 14.1 y una ejecución 14.2. Y tras los ejemplos de SQLite, encontraremos los ejemplos de MySQL 14.3 y un ejemplo interesante sobre el uso de DBI con Amazon 14.4, que deja claro que no sólo sirve para bases de datos, sino también para otras muchas cosas[16].

```

1 #!/usr/bin/perl -w
2 use strict;
3 use DBI;
4
5 my $dbh = DBI->connect( "dbi:SQLite:ejemplos.db" ) ||
6     die "Cannot connect: $DBI::errstr";
.....
9 $dbh->do( "CREATE TABLE ITEM (ID INTEGER PRIMARY KEY AUTOINCREMENT,
10     TITULO TEXT, TIPO INTEGER) );
11 $dbh->do( "INSERT INTO ITEM VALUES ( null, 'UNO', 1 ) " );
12 $dbh->do( "INSERT INTO ITEM VALUES ( null, 'DOS', 2 ) " );
13 $dbh->do( "INSERT INTO ITEM VALUES ( null, 'TRES', 3 ) " );
14 printf ("Tras Insertar Datos \n");
15 listar();
16 # A continuación se puede ver otra forma de ejecutar una Acción
17 my $sth = $dbh->prepare ( "DELETE FROM ITEM WHERE ID=2" );
18 $sth->execute ();
19 printf ("Tras Borrar Datos \n");
20 listar();
21
22 # Aquí se puede ver un estamento SQL, pero con el paso de parámetros
23 my $newtitulo = 'CUATRO';
24 my $newtitulo_quote = $dbh->quote ($newtitulo);
25 my $acambiar = 3;
26 my $acambiar_quote = $dbh->quote ($acambiar);
27
28 printf ("Tras Actualizar el Titulo del registro $acambiar por
29 $newtitulo \n");
30 $sth = $dbh->prepare ( "UPDATE ITEM SET TITULO=$newtitulo_quote
31 WHERE ID=$acambiar_quote ");
32 $sth->execute ();
33 listar();
34 $dbh->disconnect;
35
36 sub listar {
37     printf ("ID, TITULO,\tTIPO\n");
38     printf ("-----\n");
39     my $res = $dbh->selectall_arrayref( q( SELECT *
40         FROM ITEM ) );
41
42     foreach( @$res ) {
43         print "$_->[0], $_->[1], \t$_->[2]\n";
44     }

```

Cuadro 14.1: Ejemplo de Uso de DBI en el entorno de SQLite

```
ubay@REN:~/Proyecto/catalogador/bdd$ ./codDBI.pl
Tras Insertar Datos
ID, TITULO,      TIPO
-----
1, UNO,          1
2, DOS,          2
3, TRES,         3
Tras Borrar Datos
ID, TITULO,      TIPO
-----
1, UNO,          1
3, TRES,         3
Tras Actualizar el Titulo del registro 3 por CUATRO
ID, TITULO,      TIPO
-----
1, UNO,          1
3, CUATRO,       3
```

Cuadro 14.2: Ejecución del anterior Código de SQLite

```
#!/usr/bin/perl

use strict;
use DBI();

# Connect to the database.
my $dbh = DBI->connect("DBI:mysql:database=test;host=localhost",
                    "joe", "joe's password",
                    {'RaiseError' => 1});

# Drop table 'foo'. This may fail, if 'foo' doesn't exist.
# Thus we put an eval around it.
eval { $dbh->do("DROP TABLE foo") };
print "Dropping foo failed: $@\n" if $@;

# Create a new table 'foo'. This must not fail, thus we don't
# catch errors.
$dbh->do("CREATE TABLE foo (id INTEGER, name VARCHAR(20))");

# INSERT some data into 'foo'. We are using $dbh->quote() for
# quoting the name.
$dbh->do("INSERT INTO foo VALUES (1, " . $dbh->quote("Tim") . ")");

# Same thing, but using placeholders
$dbh->do("INSERT INTO foo VALUES (?, ?)", undef, 2, "Jochen");

# Now retrieve data from the table.
my $sth = $dbh->prepare("SELECT * FROM foo");
$sth->execute();
while (my $ref = $sth->fetchrow_hashref()) {
    print "Found a row: id = $ref->{'id'}, name = $ref->{'name'}\n";
}
$sth->finish();

# Disconnect from the database.
$dbh->disconnect();
```

Cuadro 14.3: Ejemplo de Uso de DBI en el entorno de MySQL[19]

```

$dbh = DBI->connect('dbi:Amazon:', $amznid, undef,
    { amzn_mode => 'books',
      amzn_locale => 'us',
      amzn_max_pages => 3
    })
    or die "Cannot connect: " . $DBI::errstr;
#
#     search for some Perl DBI books
#
$sth = $dbh->prepare("
    SELECT ASIN,
           Title,
           Publisher,
           PublicationDate,
           Author,
           SmallImageURL,
           URL,
           SalesRank,
           ListPriceAmt,
           AverageRating
    FROM Books
    WHERE MATCHES ALL('Perl', 'DBI') AND
           PublicationDate >= '2000-01-01'
    ORDER BY SalesRank DESC,
           ListPriceAmt ASC,
           AverageRating DESC");

$sth->execute or die 'Cannot execute: ' . $sth->errstr;

print join(', ', @$row), "\n"
    while $row = $sth->fetchrow_arrayref;

$dbh->disconnect;

```

Cuadro 14.4: Ejemplo de Uso de DBI en el entorno de Amazon[19]



# Parte V

## DBIx::Class



## Capítulo 15

### ¿Qué es?

Es un módulo Perl que nos facilita la posibilidad de interactuar con bases de datos, como si se tratase de una estructura típica de Perl. Más concretamente consigue traducir sentencias SQL a una API, inspirado en DBI, pero intentando ser más simple y mucho más orientado hacia su utilización en la web. Como se puede ver en la documentación el desarrollador principal del proyecto, también está implicado en el de Catalyst, un framework para el desarrollo web, que será tratado más adelante en este documento.



## Capítulo 16

### Características

La principal característica es el espíritu de crear una forma lo más sencilla posible de utilizar una base de datos en una web. A continuación se exponen algunas de las maneras..

#### 16.1. Manipulación de Relaciones

Para la creación de las relaciones en el sistema se usa la clase `DBIx::Class::Relationship`, que nos permite establecer las típicas relaciones, como son 1 a muchos, muchos a muchos o similares, de la misma forma que realizamos una llamada a una función.

#### 16.2. Definición de Estructura

Para que `DBIx` funcione correctamente se le debe indicar como está estructurada la base de datos. Pudiéndose indicar cosas como el Nombre de la Tabla, los atributos de esta y la clave principal.

#### 16.3. Búsqueda de Información en la Base de Datos

También se han definido formas de poder filtrar la información contenida en la base de datos.

Teniendo en cuenta que se puede establecer no sólo varios criterios de búsqueda, sino que también jugando con el segundo parámetro, que es opcional, la forma en que se muestra la información. Así por ejemplo se puede usar para la paginación de resultados, o también para indicar las columnas que se mostraran[12].

#### 16.4. Mostrar la Información

Posee varias funcionalidades que nos ayuda a mostrar la información de un forma clara y muy orientada hacia el entorno web. Con funciones que nos permite paginar los resultados o mostrarlos de una forma predeterminada.



## Capítulo 17

### Componentes

A continuación mostraremos los componentes de `DBIx::Class`, describiendo la utilización de estos así como el uso de estos y ejemplos cuándo sea posible.

Los componentes a utilizar son los siguientes[2]:

- `DBIx::Class::Schema` - Indicación del Esquema de la Base de Datos
- `DBIx::Class::ResultSource` - Definición de funciones sobre Tablas.
- `DBIx::Class::Relationship` - Definición de Relaciones.
- `DBIx::Class::Relationship::Base` - Definición más concreta de la Relaciones.
- `DBIx::Class::PK::Auto` - Permite el uso de la opción de numeración automática.
- `DBIx::Class::Core` - Conjunto Base de Funciones a Cargar.
- `DBIx::Class::Serialize::Storable` - Almacenaje de forma automática
- `DBIx::Class::InflateColumn` - Crea un objeto a partir de una columna de una tabla.
- `DBIx::Class::InflateColumn::DateTime` - Permite usar formatos de Fechas en una Base de Datos.
- `DBIx::Class::PK` - Permite manejar las claves primarias.
- `DBIx::Class::ResultSourceProxy::Table` - Crea un objeto que nos permite manipular la definición de una tabla.
- `DBIx::Class::AccessorGroup` - Crea conjuntos de manipuladores
- `DBIx::Class::ResultSet` - Selección y manipulación de conjuntos.

- *DBIx::Class::ResultSetColumn* - Realiza operaciones sobre columnas de un ResultSet.
- *DBIx::Class::Row* - Manipulación de Columnas.
- *DBIx::Class::Storage* - Almacenaje básico.
- *DBIx::Class::Storage::DBI* - Almacenaje usando DBI y SQL::Abstract.

### 17.1. DBIx::Class::Schema

Creas una clase para la manipulación de una base de datos basándose en un Schema. Es la forma adecuada de usar DBIx::Class y permite usar más una conexión concurrente con la clase creada.

### 17.2. DBIx::Class::Relationship

Nos permite establecer las típicas relaciones, como son 1 a muchos, muchos a muchos o similares, de la misma forma que realizamos una llamada a una función [14], [15] y [13].

#### 17.2.1. Claves Ajenas:

Se habla de clave ajena cuando se establece una relación entre dos tablas mediante el uso de una clave de una en otra. Un ejemplo sería el uso del DNI para listar cualquier informe asociado a un DNI, como por ejemplo los contratos en una empresa.

En DBIx, se hace de una forma muy sencilla, simplemente se realiza una llamada a una función, a la que se le indica, como mínimo, la tabla que exporta, el campo exportado y el campo al que corresponde en nuestra tabla final. Para que dar más claro usaré el siguiente ejemplo:

Si tenemos una tabla en forma relacional de la siguiente forma:

```
Autor (ID, .....)
```

Autor es clave ajena dependiente de la clave ID de Autor.

```
Disco(Autor, .....)
```

```
_PACKAGE_ ->belongs_to(ID => 'BDD::Autor', 'Autor');
```

Cuadro 17.1: Forma de Establecer una Relación de Clave Ajena

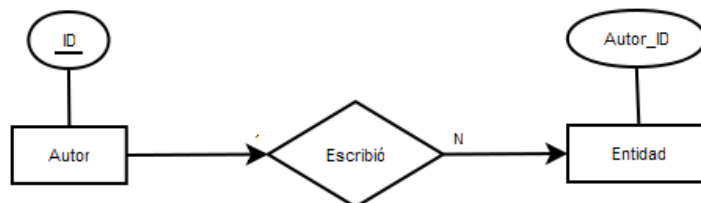
Definiríamos :

De esta manera ya se tiene definido esta relación, pero no sólo esto. Sino que tenemos otras forma de acceso

### 17.2.2. Relaciones 1 a muchos (has\_many).

En este caso estamos hablando sobre las relaciones donde un objeto tiene varios relacionados entre si. Por ejemplo, de cosas como los múltiples libros escritos por un autor.

Para declararla se llama a una función indicando como antes los mismos campos siguiendo el ejemplo anterior sería como sigue, en forma Entidad Relación:



Aquí podemos ver que la Relación “Escribió”, es del tipo 1 a Muchos. Y esto lo declararíamos de la siguiente forma:

```
__PACKAGES__->has_many(Nombre de Nuestra Relación =>
                        'Tabla que exporta', 'Clave Exportada');
```

Cuadro 17.2: Forma de Establecer una Relación de 1 a muchos

Que en nuestro ejemplo sería:

```
__PACKAGES__->has_many(map_autor => 'BDD::Autor', 'ID');
```

Cuadro 17.3: Ejemplo de Establecer una Relación de 1 a muchos

### 17.2.3. Relaciones 1 a 1.

#### Con borrado y actualización en Cascada (**might\_have**).

En este caso es una relación del tipo 1 a 1, es decir, solo puede haber una aparición de la clave exportada en la tabla ajena. Sería el caso de los ISBN de un libro, de tal manera que la relación entre un ISBN y un Título de libro ha de ser única. Y en el caso de eliminar este ISBN también desaparece la información.

```
__PACKAGES__->might_have(Nombre de Nuestra Relación =>
                        'Tabla que Exporta', 'Clave Exportada');
```

Cuadro 17.4: Forma de Establecer una Relación de 1 a 1 con Borrado en Casacada

Un ejemplo practico sería:

```
__PACKAGES__->might_have(ISBN_LIBRO => 'BBDD::Libros', 'ISBN');
```

Cuadro 17.5: Ejemplo de Establecer una Relación de 1 a 1 con Borrado en Casacada

#### Con existencia de la claves (**has\_one**).

En este caso la relación sería igual que la anterior, pero con la diferencia de que debe existir la clave en las dos tablas. Y el uso sería el siguiente:

```
__PACKAGES__->has_one(Clave => 'Tabla que exporta');
```

Cuadro 17.6: Forma de Establecer una Relación de 1 a 1 con existencia en las dos tablas

En nuestro ejemplo:

```
__PACKAGES__->has_one(ISBN => 'BBDD::Libros');
```

Cuadro 17.7: Ejemplo de Establecer una Relación de 1 a 1 con existencia en las dos tablas

#### 17.2.4. Relaciones Muchos a Muchos.

##### **many\_to\_many.**

En este caso nos encontramos ante más que una relación, con un indexado, es decir, el típico índice del final de los libros que ordenan los términos en orden alfabético y los agrupa por palabras. Ya que una sola entrada nos puede indicar varias apariciones. Un ejemplo son los actores que forman parte de una película, y que a su vez tienen más de una película, o al menos ellos esperan eso. Y se usa:

```
__PACKAGES__->many_to_many(Nombre de Nuestra Relación =>
                          'Tabla que Exporta', 'Clave Exportada');
```

Cuadro 17.8: Forma de Establecer una Relación de muchos a muchos

En nuestro ejemplo:

```
__PACKAGES__->has_many( actor_rol =>'BBDD::ActorRoles', 'role' );
__PACKAGES__->might_have(ISBN_LIBRO => 'BBDD::Libros','ISBN');
```

Cuadro 17.9: Ejemplo de Establecer una Relación de muchos a muchos

### 17.3. DBIx::Class::ResultSet

Para la manipulación de la información contenida en la Base de Datos se utiliza una serie de funciones que se encuentran en DBIx::Class::ResultSet. A continuación explicaré los métodos más interesantes:

### 17.3.1. La familia de métodos search.

Este método como se adivina nos devuelve un objeto, que se ha extraído siguiendo algún tipo de filtro que se le indique, al igual que un select. Teniendo en cuenta que se puede establecer no sólo varios criterios de búsqueda, sino que también jugando con el segundo parámetro , que es opcional, la forma en que se muestra la información. Se puede usar para la paginación de resultados, o también para indicar las columnas que se mostrarán. En search existen tres formas, esta el **search** a secas , el **search\_rd** que siempre devuelve algo aunque sea un vacío y **search\_literal**, que se le debe pasar parámetros en formato SQL.

Search - `search($condición {,%atributos});`

Search\_literal - `search_literal($fragmento_SQL, @valores).`

Find - Esta tiene la finalidad de encontrar los elementos como antes, pero sólo atendiendo a su clave principal o única .

find\_or\_new - Busca un elemento y si no existe lo crea.

find\_or\_create - Busca un elemento y si no encuentra lo crea.

Ejemplos:

```
search([ { year => 2005 }, { year => 2004 } ]);
```

```
search_literal('year = ? AND title = ?', qw/2001 Reload/);
```

```
my $cd = $schema->resultset('CD')->find_or_create({
  cdid    => 5,
  artist => 'Massive Attack',
  title   => 'Mezzanine',
  year    => 2005,
});
```

### 17.3.2. Iteradores

Estos métodos nos permiten movernos entre los elementos que se puedan haber encontrado anteriormente, con un search.

Next- Nos devuelve el siguiente elemento.

First - Devuelve el primer el elemento.

all - Devuelve todos los elementos.

### 17.3.3. Manipulando Datos

Son todos los métodos para crear, eliminar o modificar la información.

update - Actualiza un elemento.

update\_all - Actualiza todos los elementos.

update\_or\_create- Actualiza un elemento, y si no existe lo inserta.

delete - Elimina.

delete\_all - Elimina todos los elementos.

create - Inserta un elemento.



## Capítulo 18

### Ejemplo de Uso

En las siguientes líneas mostraré un ejemplo práctico de utilización de DBIx. En este ejemplo se ve una base de datos, implementada en SQLite, usada en un agente de búsqueda de información, sobre ciudades. El esquema de la base de datos se puede ver en la figura 18.1 y con esquema en SQLite que se puede ver en el cuadro 18.1.

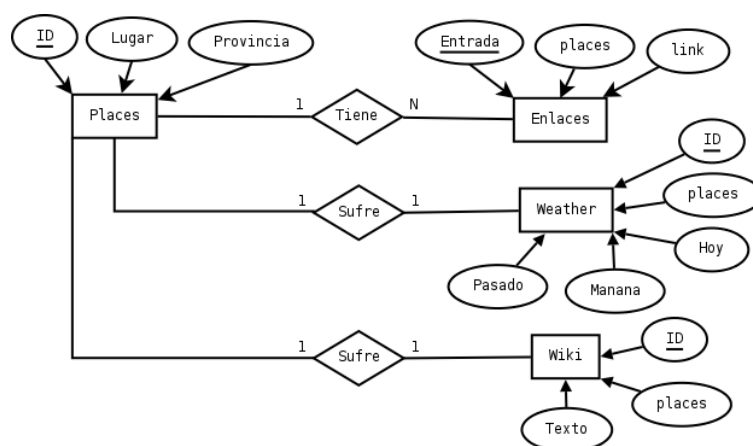


Figura 18.1: Diagrama de Base de Datos de Ejemplo

```
SQLite versión 3.4.2
Enter ".help" for instructions
sqlite> .schema
CREATE TABLE enlaces (
  entrada INTEGER PRIMARY KEY AUTOINCREMENT,
  places INTEGER NOT NULL REFERENCES places(id),
  link TEXT NOT NULL
);
CREATE TABLE places (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  lugar TEXT NOT NULL,
  provincia TEXT
);
CREATE TABLE weather (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  places TEXT INTEGER NOT NULL REFERENCES places(id),
  hoy INTEGER NOT NULL,
  manana INTEGER NOT NULL,
  pasado INTEGER NOT NULL
);
CREATE TABLE wiki (
  ID INTEGER PRIMARY KEY AUTOINCREMENT,
  places INTEGER NOT NULL REFERENCES places(id),
  texto TEXT
);
sqlite>
```

Cuadro 18.1: Esquema de Base de Datos en SQLite

A continuación se puede ver el código generado para poder representar el esquema de base de datos y en primer lugar expondré el árbol de directorios creado para manejar la base de datos, como se puede ver en el cuadro 18.2, se tiene un fichero **Main.pm**, que se puede ver en el Cuadro 18.3, donde se describe las tablas de la base de datos. Luego por cada tabla se tendrá en el directorio Main un módulo, en el que describe la estructura de esa tabla, como se puede ver para Enlaces en el cuadro 18.7, para Places en el Cuadro 18.4, para Weather en el Cuadro 18.5 y para Wiki en el Cuadro 18.6 .

```

|-- Database
|  |-- Main
|  |  |-- Enlaces.pm
|  |  |-- Places.pm
|  |  |-- Weather.pm
|  |  |-- Wiki.pm
|  |-- Main.pm
|-- basesdatos.pl
'-- datos.db

```

Cuadro 18.2: Árbol de Directorios para representar la Base de Datos

```

1 #Carga la estructura de la base de datos
2 package Database::Main;
3 use base qw/DBIx::Class::Schema/;
4 __PACKAGE__->load_classes(qw/Places Wiki Enlaces Weather/);
5
6 1;
7
8

```

Cuadro 18.3: El Fichero Main.pm

```

1 package Database::Main::Places;
2 use base qw/DBIx::Class/;
3 __PACKAGE__->load_components(qw/PK::Auto Core/);
4 __PACKAGE__->table('places');
5 __PACKAGE__->add_columns(qw/ id lugar provincia/);
6 __PACKAGE__->set_primary_key('id');
7 __PACKAGE__->has_many('wikis' => 'Database::Main::Wiki');
8 __PACKAGE__->has_many('links' => 'Database::Main::Enlaces');
9
10 1;

```

Cuadro 18.4: El fichero Places.pm

```
1 package Database::Main::Weather;
2   use base qw/DBIx::Class/;
3   __PACKAGE__->load_components(qw/PK::Auto Core/);
4   __PACKAGE__->table('weather');
5   __PACKAGE__->add_columns(qw/ id places hoy manana pasado/);
6   __PACKAGE__->set_primary_key('id');
7   __PACKAGE__->belongs_to('places' => 'Database::Main::Places');
8
9
10 1;
```

Cuadro 18.5: El Fichero Weather

```
1 package Database::Main::Wiki;
2   use base qw/DBIx::Class/;
3   __PACKAGE__->load_components(qw/PK::Auto Core/);
4   __PACKAGE__->table('wiki');
5   __PACKAGE__->add_columns(qw/ id places texto/);
6   __PACKAGE__->set_primary_key('id');
7   __PACKAGE__->belongs_to('places' => 'Database::Main::Places');
8
9
10 1;
```

Cuadro 18.6: El fichero Wiki.pm

```
1 package Database::Main::Enlaces;
2   use base qw/DBIx::Class/;
3   __PACKAGE__->load_components(qw/PK::Auto Core/);
4   __PACKAGE__->table('enlaces');
5   __PACKAGE__->add_columns(qw/ entrada places link/);
6   __PACKAGE__->set_primary_key('entrada');
7   __PACKAGE__->belongs_to('places' => 'Database::Main::Places');
8
9
10 1;
```

Cuadro 18.7: El Fichero Enlaces.pm

Y finalmente en el siguiente Cuadro 18.8 , el Cuadro 18.9 y el Cuadro 18.10 , se pueden ver algunas funciones donde se utilizan diferentes funciones del Módulo DBIx::Class::ResultSet visto en el Capitulo 17.3 . También se muestra como se realiza una conexión a la base de datos.

```
1 #!/usr/bin/perl -w
2 use strict;
3 use Database::Main;
4 use Data::Dumper;
5
6
7 my $schema = Database::Main->connect('dbi:SQLite:datos.db');
8
9 #Lista todos los lugares devolviendo un array con ellos
10
11 sub list_all_places {
12
13     my @rs = $schema->resultset('Places')->all();
14
15     return @rs
16 }
17 #devuelve el dato wiki que corresponda
18
19 sub busca_wiki {
20     my $lugar = shift;
21
22     my $rs = $schema->resultset('Wiki')->search_rs
23 ({places => $lugar});
24
25     my $wiki = $rs->first;
26     return $wiki->texto;
27 }
28 #devuelve un array con los enlaces de un dato
29
30 sub busca_enlaces {
31     my $lugar = shift;
32
33     my $rs = $schema->resultset('Enlaces')->search_rs
34 ({places => $lugar});
35
36     my @enlaces;# = map {$_->link} @rs;
37     while (my $temp = $rs->next) {
38         push (@enlaces, $temp->link);
39     }
40     return @enlaces;
41 }
```

Cuadro 18.8: El Fichero Basesdatos.pl

```

42 sub busca_weather { #devuelve el dato tiempo que corresponda
43     my $lugar = shift or die "No se ha dado clave de busqueda";
44
45     my $rs = $schema->resultset('Weather')->search_rs
46     ({places => $lugar});
47
48     my $weather = $rs->first;
49     my %salida = (HOY => $weather->hoy,
50     MANA => $weather->manana, PM => $weather->pasado);
51     #print Dumper(%salida), "\n";
52     return %salida;
53 }
54 #inserta elementos en places
55 sub inserta_place {
56
57     my $lugar = shift;
58     my $demas = shift || undef;
59
60     $schema->resultset('Places')->create ({
61         lugar=>$lugar,
62         demas=>$demas
63     });
64
65 }
66
67 #inserta elementos en wiki
68 sub inserta_wiki {
69
70     my $lugar = shift;
71     my $text = shift || "null";
72
73     $schema->resultset('Wiki')->create ({
74         places=>$lugar,
75         texto=>$text
76     });
77
78 }
79

```

Cuadro 18.9: La continuación del Fichero Basesdatos.pl

```
80 #inserta enlaces en enlace
81 sub inserta_enlace {
82
83     my $lugar = shift;
84     my $links = shift;
85
86     foreach my $link (@{$links}) {
87         $schema->resultset('Enlaces')->create ({
88             places=>$lugar,
89             link=>$link
90         });
91         print "$lugar $link\n";
92     }
93 }
94
95 }
96 #inserta elementos en wiki
97 sub inserta_weather {
98
99     my $lugar = shift or die "No se ha dado clave de busqueda";
100     my %weather =@_;
101
102
103     $schema->resultset('Weather')->find_or_create ({
104         places=> $lugar,
105         hoy => $weather{HOY},
106         manana => $weather{MANA},
107         pasado => $weather{PM}
108     });
109
110 }
111
112 inserta_wiki (2, "jfgkjdnfg dfgnkdfngdfj ldfkgjdfkngngfd\n");
113 print busca_wiki (2);
114 my %temp = (HOY => 2, MANA => 4, PM => 45);
115 print Dumper(%temp), "\n";
116 inserta_weather (2, %temp);
117 my %salida = busca_weather (2);
118 print Dumper(%salida), "\n";
```

Cuadro 18.10: La Continuación del Fichero Basesdatos.pl

**Parte VI**

**Template Toolkit**



## Capítulo 19

### ¿Qué Es?

Template Toolkit es un sistema de procesado de Template<sup>1</sup> basado en Perl, que te permite producir documentos HTML, pero también permite generar XML, PDF, y otros formatos de salida. Intenta ser sencillo de utilizar, tanto que se asegura que los templates, pueden ser mantenidos por gente que no tenga conocimientos informáticos de alto nivel, ni siquiera conocimientos sobre Perl [17].

---

<sup>1</sup>Página pre-desarrollada que es empleada para crear nuevas páginas con el mismo diseño, patrón o estilo.



## Capítulo 20

### Cosas Básicas del Lenguaje

Se debe tener en cuenta que este lenguaje está dirigido totalmente a la presentación de contenidos, pero también posee las estructuras básicas de cualquier lenguaje. Como pueden ser bucles, control de flujo y variables. Pero sin olvidar que es un lenguaje de presentación, por lo que deja las partes más complejas al lenguaje de programación que lo usa, normalmente Perl.

Todas las operaciones disponibles en este lenguaje se basan en el uso de directivas, que pueden tener este formato [% TAGS %]. Básicamente existen dos tipos de TAGS. Si el TAGS recibe un único argumento, se espera que este sea un TAG predefinido. Pero si el TAG recibe dos argumentos, esto implica que se marca un inicio y fin [4].



## Capítulo 21

### Comentarios

Los comentarios en Template Toolkit siguen el formato de Perl, es decir, *#Comentario*. Como se puede ver en el ejemplo del Cuadro 21.1

```
[% planet = 'Venus' %] #Soy un Comentario  
El planeta [% planet %] es el segundo contando desde el Sol,
```

Cuadro 21.1: Ejemplo de Comentario



## Capítulo 22

### Directivas

Por el formato de Template Toolkit, se tiene disponible un conjunto de Directivas que permite cambiar el comportamiento de este. Dentro de este conjunto de directivas se pueden encontrar de diferentes tipos.

#### 22.1. Procesado de Ficheros

En esta sección se describirá diferentes Directivas con el único fin de manipular ficheros externos. Esta manipulación será de diferentes formas, desde la inserción de la información sin modificar, a la ejecución del fichero.

##### 22.1.1. INSERT

Nos permite insertar un fichero en nuestro template, sin ser procesado. Es decir, aunque se insertara un código Template Toolkit este no sería interpretado. Para poder ser insertado este debe estar en una de la rutas indicadas a la hora de crear el objeto template, un ejemplo de esto se ve en el Cuadro 22.1. Un ejemplo de como se realizaría un Insert, se puede ver en el Cuadro 22.2

```
my $template = Template->new({
    INCLUDE_PATH => '/here:/there',
});

$template->process('myfile');
```

Cuadro 22.1: Ejemplo de como indicar las Rutas de Búsqueda de Ficheros

```
[% INSERT docs/file.txt %] #Como insertar un Fichero
[% INSERT file1.txt+file2.txt %] #Definición de varios ficheros a Insertar.

#Otra Forma de Definición del Fichero a Insertar
[% fichero = 'file3.txt' %] #Variable con el Nombre del Fichero
[% INSERT $fichero %] #Inserta el fichero indicado en la Variable.
```

Cuadro 22.2: Ejemplo de uso de Insert en Template Toolkit

### 22.1.2. INCLUDE

A diferencia de INSERT, visto en la sección 22.1.1. INCLUDE lo que inserta en el Template, es la salida de procesar el fichero indicado o el Bloque. Se debe tener en cuenta que sólo se captura la salida, es decir, un cambio en el valor de una variable dentro del INCLUDE no afectará al fichero que lo incluye, como se puede ver en el Cuadro 22.4.

Un ejemplo de uso de INCLUDE se puede ver en el Cuadro 22.3 .

```
[% myheader = 'my/misc/header' %]
[% INCLUDE myheader %] # 'myheader'
```

Cuadro 22.3: Ejemplo de como usar INCLUDE en Template Toolkit

```

[% foo = 10 %]

foo es inicialmente [% foo %]
[% INCLUDE bar %]
foo es aun [% foo %]

[% BLOCK bar %]
  foo era [% foo %]
  [% foo = 20 %]
  foo es ahora [% foo %]
[% END %]

```

Salida obtenida a partir de ejecutar lo anterior

```

foo es inicialmente 10
  foo era 10
  foo es ahora 20
foo es aun 10

```

Cuadro 22.4: Muestra del Comportamiento de las Variables en INCLUDE

### 22.1.3. PROCESS

A diferencia de INCLUDE, visto en la sección 22.1.2, en PROCESS, las modificación de una variable dentro del incluido afectará al inclusor. Como se puede ver en el Cuadro 22.9. La forma de utilizar PROCESS se puede ver en 22.8 .

```

[% myheader = 'my/misc/header' %]
[% PROCESS myheader %] # 'myheader'

```

Cuadro 22.5: Ejemplo de como usar PROCESS en Template Toolkit

```
[% foo = 10 %]  
  
foo es inicialmente [% foo %]  
[% PROCESS bar %]  
foo es aun [% foo %]  
  
[% BLOCK bar %]  
  foo era [% foo %]  
  [% foo = 20 %]  
  foo es [% foo %] ahora  
[% END %]
```

Salida obtenida a partir de ejecutar lo anterior

```
foo es inicialmente 10  
  foo era 10  
  foo es 20 ahora  
foo es 20 ahora
```

Cuadro 22.6: Muestra del Comportamiento de las Variables en PROCESS

## 22.2. WRAPPER

WRAPPER permite aplicar un Template a un Bloque terminado es [% END %]. Esto es muy utilizado a la hora de definir cabeceras que cambian según el contenido de esta, pero manteniendo características en común.

No sólo se tiene la opción de aplicar uno, sino también se puede aplicar varios como se puede ver en el Cuadro 22.7 .

Ejemppllo de uso de Wrapper, utilizando section
<pre>[% WRAPPER section   title = 'Quantum Mechanics' %]   Quantum mechanics is a very interesting subject wish   should prove easy for the layman to fully comprehend. [% END %]</pre>
section
<pre>&lt;h2&gt;[% title %]&lt;/h2&gt; &lt;p&gt;   [% content %] &lt;/p&gt;</pre>
Código HTML Generado
<pre>&lt;h2&gt;Quantum Mechanics&lt;/h2&gt; &lt;p&gt;   Quantum mechanics is a very interesting subject wish   should prove easy for the layman to fully comprehend. &lt;/p&gt;</pre>

Cuadro 22.7: Muestra del Comportamiento de WRAPPER

<pre>[% myheader = 'my/misc/header' %] [% PROCESS myheader %] # 'myheader'</pre>
--

Cuadro 22.8: Ejemplo de como usar PROCESS en Template Toolkit

```
[% foo = 10 %]

foo es inicialmente [% foo %]
[% PROCESS bar %]
foo es [% foo %] ahora

[% BLOCK bar %]
  foo era [% foo %]
  [% foo = 20 %]
  foo es [% foo %] ahora
[% END %]
```

Salida obtenida a partir de ejecutar lo anterior

```
foo es inicialmente 10
  foo era 10
  foo es ahora 20
foo es 20 ahora
```

Cuadro 22.9: Muestra del Comportamiento de las Variables en PROCESS

### 22.3. BLOCK

En las secciones anteriores se ha nombrado varias veces el término **Bloque**, pero no se había definido. La definición es sencilla ya que un bloque son un conjunto de líneas que deben ser procesadas, por INSERT, WRAPPER, etc.

Tiene diferentes forma de utilizar pero la más sencilla se muestra en el Cuadro 22.10 .

Forma básica de Declarar y Usar un Bloque
<pre> [% BLOCK tabrow %] &lt;tr&gt;   &lt;td&gt;[% name %]&lt;td&gt;   &lt;td&gt;[% email %]&lt;/td&gt; &lt;/tr&gt; [% END %]  &lt;table&gt;   [% PROCESS tabrow name='Fred' email='fred@nowhere.com' %]   [% PROCESS tabrow name='Alan' email='alan@nowhere.com' %] &lt;/table&gt; </pre>
Capturando contenido con un Bloque Anónimo
<pre> [% julius = BLOCK %]   And Caesar's spirit, ranging for revenge,   With Ate by his side come hot from hell,   Shall in these confines with a monarch's voice   Cry 'Havoc', and let slip the dogs of war;   That this foul deed shall smell above the earth   With carrion men, groaning for burial. [% END %] </pre>

Cuadro 22.10: Formas de Utilización de Block

## 22.4. Filtros y Plugins

### 22.4.1. FILTER

Esta directiva permite realizar un post-proceso a un Bloque, como por ejemplo el HTML, que permite eliminar todos los elementos propios de este lenguaje. En el Cuadro 22.11 se puede ver un ejemplo de uso. En la tabla 22.12 se muestra una lista de algunos filtros.

Filtro	Acción
<b>Collapse</b>	Cambia múltiples espacios por un único espacio.
<b>eval / evaltt</b>	Evalua el bloque como si fuera código Tempalte Toolkit
<b>format(fmt)</b>	Dato el formato fmt se lo aplica al bloque linea a linea.
<b>html</b>	Cambia los caracteres extraños por codificación estandar HTML
<b>latex(outputType)</b>	Dado un bloque latex devuelve un PDF, DVI o PS

Cuadro 22.12: Algunos Filtros interesantes incluidos en Template Toolkit.

Ejemplo de Utilización de Filter
<pre>[% FILTER html %]   HTML text may have &lt; and &gt; characters embedded   which you want converted to the correct HTML entities. [% END %]</pre>
Salida esperada al usar el Filtro
<pre>HTML text may have &amp;lt; and &amp;gt; characters embedded which you want converted to the correct HTML entities.</pre>

Cuadro 22.11: Formas de Utilización de Filter

### 22.4.2. USE

Permite cargar Plugins, que tienen múltiples plugins que permite realizar múltiples acciones con bloques. El uso de USE se puede ver en el Cuadro 22.13 y en la Tabla 22.14 se listan algunos de los Plugins actuales.

Plugin	Acción
<b>Autoformat</b>	Permite aplicar formatos a un bloque.
<b>CGI</b>	Permite manipular las variables típicas de Formularios
<b>DBI</b>	Interface a Perl DBI, explicado en la sección IV.
<b>GD</b>	Acceso a las librerías GD de manipulación de gráficos.
<b>XML::RSS</b>	Creación de RSS.

Cuadro 22.14: Algunos Plugins interesantes incluidos en Template Toolkit.

Forma de cargar un Plugin
<code>[% USE myplugin %]</code>
Ejemplo de Carga de un Plugin
<pre>[% USE CGI    %]      # =&gt; Template::Plugin::CGI [% USE Cgi    %]      # =&gt; Template::Plugin::CGI [% USE cgi    %]      # =&gt; Template::Plugin::CGI</pre>

Cuadro 22.13: Forma de uso de USE



## Capítulo 23

### Variables

En Template Toolkit también existen las variables, pero básicamente existen escalares, listas y hashes. Todas estas tienen en común la existencia de métodos virtuales, que nos permite manipular la información contenida en ellos de una forma más sencilla.

#### 23.1. Escalares

Son el tipo más sencillo de entender pero también son las más usadas, la idea es almacenar en ellas elementos singulares. Es decir, cosas como números, cadenas de texto, etc.

El uso sería el visto en el Cuadro 23.1 :

```
[% planet = 'Venus' %]  
El planeta [% planet %] es el segundo contando desde el Sol,
```

Cuadro 23.1: Uso de Variable Escalar en Template Toolkit

#### 23.2. Listas

Estas variables son lista de elementos almacenadas, como los vectores en otros lenguajes. Se definen mediante el uso de [...]. Y para acceder a la información se usa el punto. En el ejemplo del Cuadro 23.2 se ve más claramente.

```
[% planetas = ['Venus', 'Marte', 'Jupiter'] %]

Acceso:

[% planetas.0 %] #Para poder acceder al valor del elemento 0

[% index = 1 %]
[% planetas.$index %] #Otra forma de acceder a la información
```

Cuadro 23.2: Uso de Listas en Template Toolkit

### 23.3. Hashes

Estas variables son como las listas, pero en lugar de acceder a la información mediante su orden, se hace mediante una clave. Esto se define mediante el uso de `{...}` . Como se puede ver en el Cuadro 23.3 .

```
[% planetas = {
    primero => 'Venus',
    segundo => 'Marte',
    tercero => 'Jupiter'
} %]

Acceso:

[% planetas.primerero %] #Para poder acceder al valor del elemento
#primerero

[% key = 'primerero' %]
[% planetas.$key %] #Otra forma de acceder a la información
```

Cuadro 23.3: Uso de Listas en Template Toolkit

## 23.4. Métodos Virtuales

Estos métodos se tienen con el fin de manipular las variables de una forma más eficaz. Consiguiendo realizar operaciones complicadas de una forma más sencilla, como obtener el tamaño de un escalar, siempre y cuando sea una cadena, u obtener una lista ordenada.

### 23.4.1. Métodos Escalares

Estos son los métodos que se pueden aplicar a los escalares.

**chunk(size)** Trozea una cadena, en un tamaño máximo indicado por Size

**defined** Devuelve True si el valor está definido. Aunque el escalar este vacío.

**hash** Convierte un Escalar en un Hash, donde la clave es “value”.

**length** Devuelve el número de caracteres de una cadena.

**match(pattern)** Permite buscar pattern, que debe ser una expresión regular, en el escalar.

**repeat(n)** Repite el elemento original n veces.

**replace(search, replace)** Busca search y lo reemplaza por replace. Siendo ambas expresiones regulares.

### 23.4.2. Métodos referidos a Listas

Estos son los métodos que se pueden utilizar con Listas, elementos de Listas o cualquier elemento bendecido con listas.

**first(n)** Devuelve los n primeros elementos de una lista.

**grep(pattern)** Retorna una lista con todo elemento de una lista que cumpla con el pattern que debe ser una expresión regular.

**join(delimitador)** Retorna una cadena con los elementos que componen una lista. Los elementos están separados por el delimitador indicado.

**last(n)** Retorna los n finales elementos de una lista.

**max** Indica el índice del elemento con el máximo valor.

**merge(list)** Crea una lista a partir de las listas indicadas por list. Y la lista en la que se llama este método.

**pop** Elimina el último elemento y lo devuelve.

**reverse** Retorna una referencia a una nueva lista, con los mismo elementos pero en orden inverso.

**shift** Elimina el primer elemento. Y lo retorna.

**size** Nos indica el número de elementos en una lista.

**slice(from, to)** Retorna los elementos que se sitúen entre from y to.

**sort, nsort** sort retorna la lista ordenada de forma alfabética y nsort de forma numérica.

**splice(offset, n, lista)** Nos permite eliminar los n elementos, devolviendolos; desde offset retornandolos en la lista indicada.

**unique** Elimina los elementos repetidos en la lista indicada.

**unshift(item)** Inserta item en el comienzo de la lista.

**push(item)** Inserta item al final de la lista.

### 23.4.3. Métodos referidos a Hash

Estos son los métodos referidos a Hashes y cualquier estructura bendecida como un hash.

**defined(key)** Indica con true si key está definido en el hash.

**each** Devuelve en forma de lista el hash, con la siguiente forma. Key1, valor1, key2, valor2.

**exists(key)** Indica si key está en el hash, aunque el valor asociado no este definido.

**import(hash)** Nos permite incrustar un hash dentro de otro.

**item(key)** Otra forma de acceder a la información en un hash.

**keys** Nos devuelve una lista con las claves del hash.

**list** Nos devuelve una referencia a una lista. Si se usa esta forma `list('keys')`, la lista estará formada por las claves del hash. Si se indica así `list('values')`, se obtiene una lista con los valores del hash. O si se hace de esta forma `list('each')`, la lista estará formada por clave, valor. Y si no indica algo se obtiene una lista de hashes donde cada hash está formado por una entrada de la anterior.

**size** Nos devuelve el tamaño del hash.

**sort, nsort** Nos devuelve una lista ordenada por las claves en orden alfabético al usar **sort**. Y si se usa **nsort** se ordena de forma numérica los valores del hash.

**values** Devuelve una lista con los valores de un hash.



## Capítulo 24

### Control de Flujo

Al igual que otros lenguajes también se pueden encontrar estructuras para controlar la ejecución.

#### 24.1. Sentencias Condicionales

Básicamente si el lector conoce las sentencias condicionales en otro lenguaje entenderá como funciona en Template Toolkit. Ya que también encontrará el IF, ELSIF, ELSE y UNLEES.

##### 24.1.1. IF, ELSIF, ELSE y UNLESS

La sintaxis usada es la siguiente:

```
[% IF test %]  
Acción  
[% ELSIF test %]  
Acción  
[% ELSE %]  
Acción  
[% END %]
```

Cuadro 24.1: Uso de sentencias Condicionales IF Template Toolkit

##### 24.1.2. SWITCH y CASE

Su sintaxis es la que se puede ver en el Cuadro 24.2:

#### 24.2. Bucles

En este apartado se muestra las típicas sentencias de bucles.

```

[% SWITCH variable %]
[% CASE Valor1 %]
Acción.
[% CASE Valor2 %]
Acción.
[% CASE %] # Acción por defecto
Acción.
[% END %]

```

Cuadro 24.2: Uso de sentencias Condicionales SWITCH Template Toolkit

### 24.2.1. FOREACH

Un FOREACH, nos permite iterar sobre todos y cada uno de los elementos que forman una lista o un hash. Así mismo al usar FOREACH, se crea de forma implícita un objeto *iterador*, que nos proporciona información adicional que se indica en el cuadro 24.4 .

La sintaxis básica del FOREACH es la siguiente, vista en el cuadro 24.3:

```

[% FOREACH item IN list %]
Acciones
[% END %]

o También

[%FOREACH item = list %]
Acciones.
[% END %]

```

Cuadro 24.3: Sintaxis de FOREACH en Template Toolkit

### 24.2.2. WHILE

Básicamente es igual a cualquier otro WHILE, pero con la diferencia de sintaxis y que posee un número máximo de iteraciones definido por el sistema , en nuestro caso 1000. Pero como todo lo existente, se puede cambiar su comportamiento, en

**size** Tamaño de la lista tratada.

**max** El Índice máximo.

**index** Nos indica el índice del item actual.

**count** Nos indica el índice actual, pero iniciando la cuenta desde 1.

**first** Nos indica con TRUE si el iterador se encuentra en la primera posición.

**last** Al igual que **first**, pero con respecto a la última posición.

**prev** Nos retorna el item anterior, y en caso de estar en el primero, retornará Undef.

**next** Al igual que **prev**, pero con respecto al siguiente item.

```
[% FOREACH item IN items %]
    [% IF loop.first %] #loop es el objeto iterador
                        <ul>
    [% END %]
    .....
    .....
[% END %]
```

Cuadro 24.4: Posibilidades de iterator Template Toolkit

este caso manipulando la variable de paquete siguiente, `TemplateDirectiveWHILE_MAX`. La sintaxis básica se puede ver en el cuadro 24.5 .

```
[% WHILE (test) %]  
Acciones  
[% END %]
```

Ejemplo de uso

```
[% total = 0 %]  
[% WHILE (total <= 100) %]  
Total: [% total]  
[% total = total + 1 %]  
[% END %]
```

Cuadro 24.5: Sintaxis de WHILE en Template Toolkit

## Capítulo 25

### Generando Páginas Dinámicas

Aunque se pueden generar contenidos de forma estática, pero como durante todo el documento se ha hablado de la utilización de bases de datos, que es básicamente dinámico. Por esta razón intentaré a continuación mostrar como realizar nuestra primera página, en este caso, un Hola Mundo, que se verá en el cuadro 25.1.

A continuación explicaré brevemente que se puede ver en el Cuadro 25.1. Primero es importante entender como se pasa información al Template desde el Script. Para ello sólo tenemos que declarar el objeto **Template**. Tras esto debemos declarar un Hash, con las variables a utilizar. Después se le debe pasar a *process*, que enviará la variables definidas, al Template indicado por `$file`.

Código Perl para enviar información
<pre>#!/usr/bin/perl use strict; use warnings; use Template;  my \$file = 'src/greeting.html'; my \$vars = {     message =&gt; "Hola Mundo\n" };  my \$template = Template-&gt;new();  \$template-&gt;process(\$file, \$vars)        die "Template process failed: ", \$template-&gt;error(), "\n";</pre>
Template para Generar el Cuerpo
<pre>[% INCLUDE header %]  &lt;p&gt; [% message %] &lt;\p&gt;  [% INCLUDE footer %]</pre>
Fichero HTML generado
<pre>&lt;html&gt;   &lt;head&gt;     &lt;title&gt;Hola Mundo v1&lt;/title&gt;   &lt;/head&gt;   &lt;body&gt;      &lt;p&gt; Hola Mundo     &lt;\p&gt;   &lt;/body&gt; &lt;/html&gt;</pre>

Cuadro 25.1: El Hola Mundo con Template Toolkit Versión 1

# Parte VII

## Catalyst



## Capítulo 26

### Introducción

Catalyst [1] es un framework web de código abierto basado en Perl, que sigue la arquitectura Modelo-Vista-controlador (MVC), que nos permite crear diversos tipos de aplicaciones Web. Estando muy influenciado por otros frameworks como son `RubyonRails`, `Maypole` y `Spring`. Su distribución se realiza mediante la plataforma CPAN, dándonos una idea de la calidad del código desarrollado.

Tras dar una definición vamos a intentar entender mejor lo dicho anteriormente, lo primero es tener claro que es un framework web, no es más que un conjunto de herramientas que nos permite diseñar aplicaciones web de una forma más sencilla ahorrando la repetición de ciertas fases comunes a toda aplicación. Otro concepto que se debe tener claro es el de **Modelo-Vista-Controlador**<sup>1</sup>, cuando se habla de esto se quiere decir que cuando se refiere a las reglas de transformación de la información de una aplicación se habla del Controlador, cuando se trata como se ve esta aplicación se habla de Vista y cuando se comenta las formas de acceso a la Base de Datos hablamos del Modelo. Para concretar Catalyst en una herramienta que nos permitirá crear diferentes tipos de aplicaciones Web, pero sin tener que repetir pasos, sino que todo ese tipo de características típicas en una aplicación serán suministradas por el framework.



Figura 26.1: Logotipo del Framework Catalyst.

---

<sup>1</sup>Referencia Web [http://es.wikipedia.org/wiki/Modelo\\_Vista\\_Controlador](http://es.wikipedia.org/wiki/Modelo_Vista_Controlador)

## 26.1. Filosofía

La filosofía de este framework es la de **No Rehacer tú Trabajo**, es decir cuando hagas algo no lo repitas de nuevo reutilízalo y en este sentido Catalyst [1] permite utilizar prácticamente la totalidad de módulos disponibles en CPAN o los que el desarrollador pudiera haber creado anteriormente.

Para cubrir los diferentes aspectos de la arquitectura MVC, usa diferentes módulos ya existentes en CPAN pero en ningún caso obligando a utilizar alguno en concreto, así para :

**Modelo** , se puede utilizar DBIx::Class, Plucene, Net::LDAP o cualquier otro modelo disponible.

**Vista** , se puede usar Template Toolkit o HTML::Template.

**Controlador** , debe ser escrito por el desarrollador de la aplicación pero para ayudar existen múltiples plugins en CPAN.

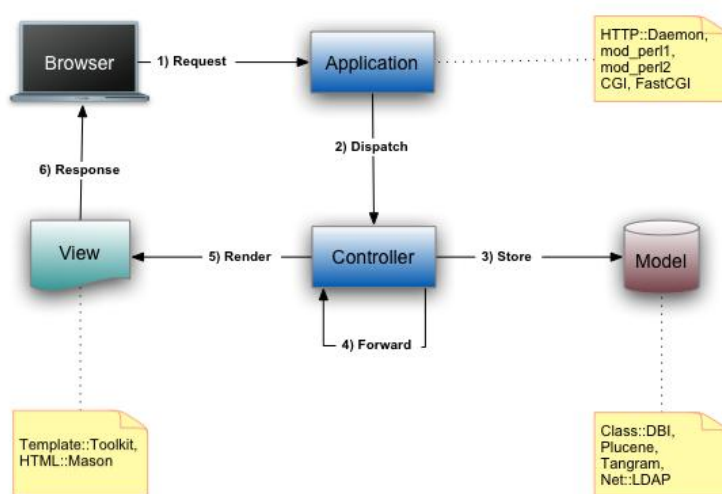


Figura 26.2: Flujo de Trabajo de Catalyst

## 26.2. Servidores Webs soportados

El más conocido es Apache con FastCGI o mod.Perl, pero en general cualquiera que soporte CGI o FastCGI funciona. Pero también se puede utilizar el integrado en Catalyst para el desarrollo. Más adelante se explicará como poder utilizar nuestro trabajo en Apache [21], .

### 26.2.1. Pasando de Desarrollo a Producción

En las siguiente páginas se indicarán formas para poder crear nuestra aplicación. Pero todo el trabajo no es sólo desarrollar también se debe poner en producción cosa que se intentará explicar como hacer en las siguientes líneas.

#### FastCGI

Simplemente con tener disponible FastCGI en nuestro sistema podremos configurar nuestra aplicación, como se muestra a continuación.en el Cuadro 26.1

Configuración en el Servidor Apache
<pre>FastCgiServer /src/myapp/script/myapp_fastcgi.pl -processor 3 Alias /myapp/ /src/myapp/script/myapp_fastcgi.pl/</pre>
<p>En la primera línea se esta configurando la opción de que sólo se ejecuten 3 procesos de nuestra aplicación y en la segunda se establece un Alias de nuestra aplicación.</p>

Cuadro 26.1: Ejemplo de FastCGI

#### mod\_perl

Hasta hace un tiempo la única forma que teníamos para realizar aplicaciones eficientes era el uso de `mod_perl` [11], pero para poder utilizar junto con Catalyst, debemos tener disponible en nuestro sistema el módulo `Catalyst::Engine::Apache`, la forma de configurar nuestra aplicación deberemos hacerlo como se ve en el cuadro 26.2 .

#### Problemas y Soluciones

Por las referencias tanto si se utiliza FastCGI o `mod_perl` , se tiene el mismo problema, que básicamente se tiene una instancia de nuestra aplicación por cada petición, cosa que FastCGI intenta resolver con el parámetro de procesos máximos, pero incluso así se producen problemas de rendimiento. Para intentar solucionar esto se plantea la opción de utilizar los servidores de desarrollo incluidos en Catalyst, pero balanceando la carga entre ellos, para lo cual se utiliza `Perlbal`

**Configuración en el Servidor Apache 2.x**

```
PerlSwitcher -I /myapp/lib
PerlModule Myapp
<Location /myapp> #Aquí se debe especificar
SetHandler modperl # el nombre de nuestra aplicación en el servidor
PerlResponseHandler Myapp
<\ Location>
```

La configuración es sencilla, ya que simplemente se crea una localización en nuestro servidor Web y se indica de donde obtener el sistema.

Cuadro 26.2: Ejemplo de FastCGI

(<http://www.danga.com/perlbal/>). Para ello simplemente se debería instalar PerlBal, luego tener arrancado una serie de servidores en diferentes puertos y estos estar dados de alta en PerlBal.

**26.3. Alternativas a Catalyst**

En esta sección quiero dejar claro que no sólo existe Catalyst y en cierta medida ayudar al lector en la búsqueda de alternativas que se pueden ver en los Cuadro 26.3 y 26.4 [24].

Proyecto	Versión Actual	Lenguaje	Licencia
Ajile	1.2.1	JavaScript	MPL 1.1 / LGPL 2.1 / GPL 2.0
Akelos	0.8	PHP	LGPL
Apache Cocoon	2.1.11	Java	Apache
Apache Struts	2.0.11	Java	Apache
AppFuse	.2.0	Java	.CAL
ASP Xtreme Evolution	1.0	VBScript	LGPL
Aranea MVC	1.0.10	Java	Apache
BFC	7.40	ASP.NET	Base One EU- LA
CakePHP	1.1.18.5850	PHP	MIT
Camping	1.5	Ruby	MIT
Canvas Framework	.	PHP	.
Catalyst	5.7011	Perl	GPL/Artistic
CherryPy	3.0.2	Python	BSD
Code Igniter	1.6.1	PHP	Apache/BSD- style osl
ColdBox Framework	2.5.1	ColdFusion	Apache 2.0 osl
ColdSpring	.	ColdFusion	.
CSLA	.	ASP.NET	.
DIY Framework	.	PHP	.
Django	0.96	Python	BSD
DotNetNuke	4.8.0	ASP.NET	BSD
Drupal	6.0	PHP	GPL
eZ Components	2007.2	PHP	BSD
Flex	2.0	ActionScript 3	GPL
FUSE		PHP	.
Fusebox	5.1	ColdFusion	Apache
Google Web Toolkit	.	Java	Apache
Grails	1.0	Groovy (JVM)	Apache
Hamlets	1.4	Java	BSD
Helma	1.6	Javascript	Apache
Horde	.	PHP	
Interchange	.	Perl	
ItsNat	.	Java	AGPLv3
IT Mill Toolkit	.	Java	
JavaServer Faces	.	Java	
JBoss Seam	2.0.0 GA	Java	LGPL
jZeno	1.0.36	Java	LGPL
Kohana	2.1.1	PHP	BSD
Lift	0.6.0	Scala (JVM)	Apache
Mason	.	Perl	

Cuadro 26.3: Primer Cuadro de alternativas a Catalyst

Proyecto	Versión Actual	Lenguaje	Licencia
Maypole	.	Perl	
Mach-II	1.5	ColdFusion	Apache
Model-Glue	2.0	ColdFusion	Apache
MonoRail	1.0 RC3	ASP.NET	Apache
Morfik	1.3.1.18	Basic,Pascal,Java,C#	Commercial
Nitro	0.41	Ruby	BSD
onTap	3.0	ColdFusion	BSD
OpenACS	5.3.2	Tcl	GPL
OpenLaszlo	.	Java	
OpenXava	2.2.4	Java	LGPL
Play!	r160	Java	GPL
Pal]	0.1	PHP	Apache
PRADO	3.1.1	PHP	BSD
Pylons (web framework)	0.9.6	Python	BSD
Qcodo	0.3.32	PHP	MIT License
Reasonable Server Faces (RSF)	.	Java	
RIFE	1.6.2	Java	Apache
Ruby on Rails	2.0.2	Ruby	MIT/Ruby
Seaside	2.8	Smalltalk	MIT License
Shale Framework (software)	.	Java	
Simplicity PHP framework	.	PHP	GPL
SilverStripe / Sapphire	2.2.1	PHP	BSD
SmartClient	.	Java	
Spring Framework	2.5	Java	Apache
Stripes	1.4.3	Java	LGPL
Symfony	1.0.10	PHP	MIT
Tapestry	4.1.3/5.0.7	Java	Apache
ThinWire	1.2	Java	GPL
TurboGears	1.0.1	Python	MIT License, LGPL
Web2py	1.10	Python	GPL .
WebObjects	5.4	Java	Proprietary
WebWork	.	Java	
Wicket framework	1.2.7	Java	Apache
Widgetplus framework	0.0.8	JavaScript	GPL
Zend Framework	1.5	PHP	BSD
ZK Framework	3.0.1	Java	GPL
Zoop Framework	1.2	PHP	ZPL
Zope2	2.10	Python	ZPL
Zope3	3.3	Python	ZPL
ztemplates	.	Java	Apache
Lion framework	.	PHP	MIT License,

Cuadro 26.4: Segundo Cuadro de alternativas a Catalyst

## Capítulo 27

### Primeros Pasos en Catalyst

En este capítulo se tiene la intención de mostrar al usuario como dar sus primeros pasos en Catalyst, mostrando de la forma más práctica posible, como crear una aplicación web como base de otra mucho más complicada.

#### 27.1. Instalando Catalyst

En esta sección se muestra como instalar el paquete en el sistema, mediante las diferentes formas existentes.

##### 27.1.1. Instalándolo como un paquete Debian

Como otros muchos otros módulos Perl, también se encuentra como paquete Debian en muchos de los repositorios existentes. Como se puede ver en el cuadro 27.1.

```
ubay@REN:~/home/ubay/ sudo apt-get install libcatalyst-perl
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
 libcatalyst-perl
0 actualizados, 1 se instalarán, 0 para eliminar y 0 no actualizados.
```

Cuadro 27.1: Instalando Paquete Debian [21]

##### 27.1.2. Instalando mediante CPAN

Para poder instalar Catalyst desde CPAN, en el caso de que no este disponible en nuestra distribución, se podrá instalar fácilmente pero con la desventaja que se necesitaran herramientas como un compilador (gcc), make y otras utilidades

destinadas al desarrollo de aplicaciones. El comando necesario se puede ver en el Cuadro 27.2 .

```
ubay@REN:~/home/ubay/ sudo cpan Catalyst::Runtime Catalyst::Devel
```

Cuadro 27.2: Instalando Mediante CPAN

## 27.2. Creando lo Básico de Catalyst

Esta sección tiene la misión de introducir al lector a los pasos básicos para crear una aplicación, es decir, básicamente se busca tener claro como crear el esqueleto de la aplicación, como configurar el acceso a bases de datos y mostrar información. Para mostrar los pasos se utilizará un ejemplo que está basado en lo visto en la Parte II más concretamente el Capítulo 5 , pero simplificando el esquema visto en la Figura 5.1. Básicamente este esquema sólo nos permite manejar el título del elemento, el tipo de elemento, su localización e información sobre el elemento. Un diagrama se puede ver en la Figura 27.1 , el ejemplo lo orientaremos hacia el manejo de una colección de Películas, pero eso será tratado más adelante.

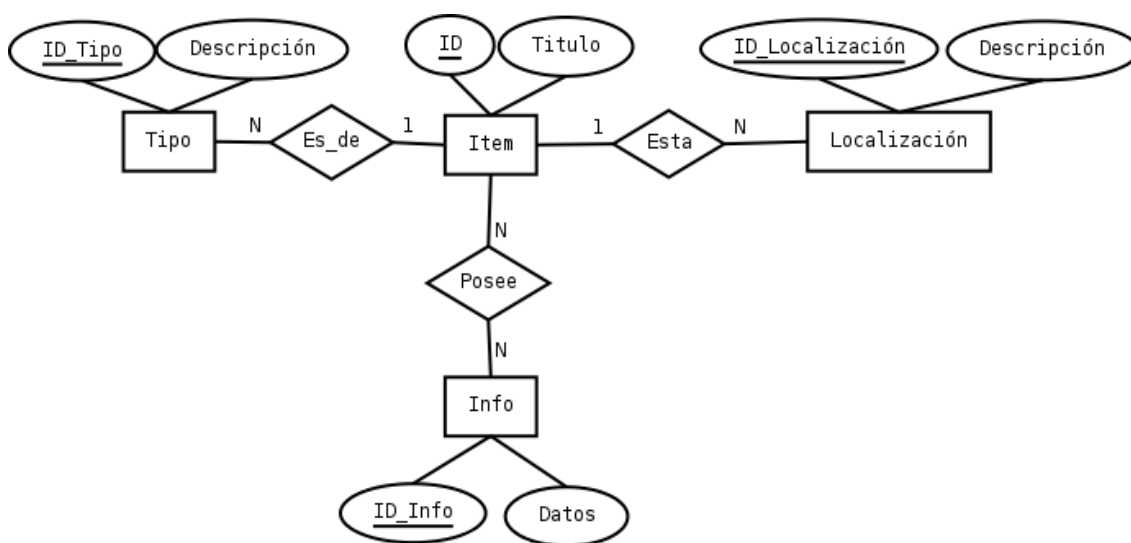


Figura 27.1: Esquema Resumen del Ejemplo para Catalyst

### 27.2.1. Los Primeros Pasos

```
davioth@kathia:~/Proyecto/catalogador/catalyst$ catalyst.pl Ejemplo
created "Ejemplo"
created "Ejemplo/script"
created "Ejemplo/lib"
created "Ejemplo/root"
created "Ejemplo/root/static"
created "Ejemplo/root/static/images"
created "Ejemplo/t"
created "Ejemplo/lib/Ejemplo"
created "Ejemplo/lib/Ejemplo/Model"
created "Ejemplo/lib/Ejemplo/View"
created "Ejemplo/lib/Ejemplo/Controller"
created "Ejemplo/ejemplo.yml"
created "Ejemplo/lib/Ejemplo.pm"
created "Ejemplo/lib/Ejemplo/Controller/Root.pm"
created "Ejemplo/README"
created "Ejemplo/Changes"
created "Ejemplo/t/01app.t"
created "Ejemplo/t/02pod.t"
created "Ejemplo/t/03podcoverage.t"
created "Ejemplo/root/static/images/catalyst_logo.png"
created "Ejemplo/root/static/images/btn_120x50_built.png"
created "Ejemplo/root/static/images/btn_120x50_built_shadow.png"
created "Ejemplo/root/static/images/btn_120x50_powered.png"
created "Ejemplo/root/static/images/btn_120x50_powered_shadow.png"
created "Ejemplo/root/static/images/btn_88x31_built.png"
created "Ejemplo/root/static/images/btn_88x31_built_shadow.png"
created "Ejemplo/root/static/images/btn_88x31_powered.png"
created "Ejemplo/root/static/images/btn_88x31_powered_shadow.png"
created "Ejemplo/root/favicon.ico"
created "Ejemplo/Makefile.PL"
created "Ejemplo/script/ejemplo_cgi.pl"
created "Ejemplo/script/ejemplo_fastcgi.pl"
created "Ejemplo/script/ejemplo_server.pl"
created "Ejemplo/script/ejemplo_test.pl"
created "Ejemplo/script/ejemplo_create.pl"
davioth@kathia:~/Proyecto/catalogador/catalyst$
```

Cuadro 27.3: Creación de Esqueleto de Catalyst

En nuestro ejemplo lo primero que debemos hacer es crear un esqueleto que nos permitirá seguir adelante en nuestro ejemplo, para ellos podemos utilizar las herramientas propias de Catalyst, más concretamente la orden[21]

```
ubay@ren: /home/ubay/catalyst.pl Ejemplo ,
```

*catalyst.pl* es la herramienta que crea el esqueleto mientras que *Ejemplo* es el nombre de nuestra aplicación. En el Cuadro 27.3 veremos la ejecución y en el Cuadro 27.4 se puede ver el Esqueleto inicial, que se explicará más concretamente más adelante. En la Figura 27.2 se puede ver la Web que el sistema crea de forma predeterminada, así mismo como la dirección que en caso de no indicar lo contrario es *http://localhost:3000* . En el Cuadro 27.6 se puede ver la petición y la respuesta a la acción de introducir la dirección anterior en nuestro navegador.

Lo que se puede ver en el Cuadro 27.4, básicamente es una estructura de directorios, donde los más importantes son **lib**, que es el centro de nuestra aplicación es también donde se puede encontrar el Modelo, la Vista y el Controlador o mejor dicho donde se deben establecer todos estos procesos. En el directorio **script**, donde encontraremos diferentes herramientas que nos ayudan a la hora de la creación de la aplicación. Y finalmente **static**, que es lugar donde se debe colocar todo el contenido estático que deseemos utilizar. Así mismo el **root**, es la raíz de nuestra aplicación.

```

Ejemplo
|-- Changes
|-- Makefile.PL
|-- README
|-- ejemplo.yml
|-- lib
|   |-- Ejemplo
|   |   |-- Controller
|   |   |   '-- Root.pm
|   |   |-- Model
|   |   '-- View
|   '-- Ejemplo.pm
|-- root
|   |-- favicon.ico
|   '-- static
|       '-- images
|           |-- btn_120x50_built.png
|           |-- btn_120x50_built_shadow.png
|           |-- btn_120x50_powered.png
|           |-- btn_120x50_powered_shadow.png
|           |-- btn_88x31_built.png
|           |-- btn_88x31_built_shadow.png
|           |-- btn_88x31_powered.png
|           |-- btn_88x31_powered_shadow.png
|           '-- catalyst_logo.png
|-- script
|   |-- ejemplo_cgi.pl
|   |-- ejemplo_create.pl
|   |-- ejemplo_fastcgi.pl
|   |-- ejemplo_server.pl
|   '-- ejemplo_test.pl
'-- t
    |-- 01app.t
    |-- 02pod.t
    '-- 03podcoverage.t

```

Cuadro 27.4: Esqueleto Inicial de Catalyst

```
davioth@kathia:~/.../Ejemplo$ script/ejemplo_server.pl
[debug] Debug messages enabled
[debug] Loaded plugins:
-----
| Catalyst::Plugin::ConfigLoader 0.17 |
| Catalyst::Plugin::Static::Simple 0.19 |
-----
[debug] Loaded dispatcher "Catalyst::Dispatcher"
[debug] Loaded engine "Catalyst::Engine::HTTP"
[debug] Found home "~/.../Ejemplo"
[debug] Loaded Config "~/.../Ejemplo/ejemplo.yml"
[debug] Loaded components:
-----+-----
| Class | Type |
+-----+-----+
| Ejemplo::Controller::Root | instance |
-----+-----
[debug] Loaded Private actions:
-----+-----+-----
| Private | Class | Method |
+-----+-----+-----+
| /default | Ejemplo::Controller::Root | default |
| /end | Ejemplo::Controller::Root | end |
-----+-----+-----
[info] Ejemplo powered by Catalyst 5.7010
You can connect to your server at http://kathia:3000
```

Cuadro 27.5: Ejecución Del Servidor Incluido en Catalyst, la primera Vez

### 27.3. Haciendo un Hola Mundo

Antes de continuar con cosas más interesantes, primero vamos a realizar el típico Hola mundo de todo lenguaje de programación, mediante el uso de un ejemplo[18].

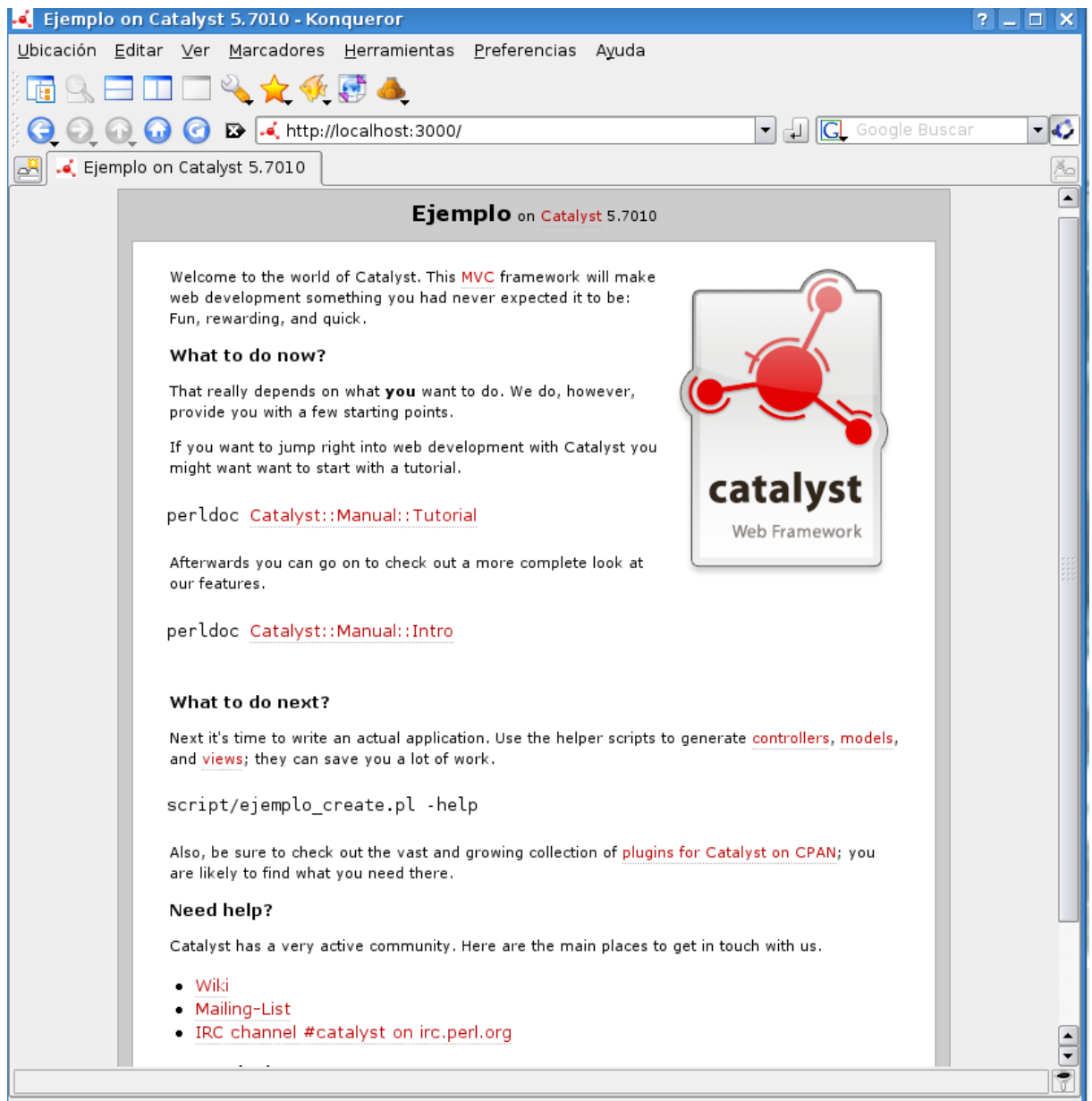


Figura 27.2: Web Por defecto Generada por Catalyst

```
[info] *** Request 1 (0.021/s) [17832] [Mon May 12 21:35:31 2008] ***
[debug] "GET" request for "/" from "127.0.0.1"
[info] Request took 0.055039s (18.169/s)
.-----+-----+
| Action                                     | Time           |
+-----+-----+
| /default                                  | 0.000367s     |
| /end                                       | 0.000771s     |
'-----+-----'
```

Cuadro 27.6: Petición de Web Inicial de Catalyst

### 27.3.1. Añadiendo una Vista

Lo primero que debemos hacer es añadir una forma de mostrar el contenido, es decir, una Vista. En nuestro caso usaremos para ello Template Toolkit, que ya fue introducido anteriormente <sup>1</sup>, para poder hacer esto deberemos primero disponer del módulo para ello, una forma sencilla es instalar el módulo Catalyst como se ve a continuación

```
cpan -i Catalyst::View::TT
,esto nos instalará todo lo necesario.
```

Tras instalar en nuestro sistema lo necesario pasaremos a crear lo necesario para poder tener nuestra vista. Para esto se utiliza un script disponible en el esqueleto creado anteriormente, para hacer esto usaremos el siguiente comando

```
script/ejemplo_create.pl view TT TTSite
```

, esto creará todo lo necesario para nuestro motor de vista. La sintaxis es sencilla ya que **view** indica vista, **TT** que se usará Template Toolkit y **TTSite** el nombre de esta. En el Cuadro 27.7 se puede ver una ejecución de este y la estructura creada en la cual se colocaran nuestros Templates, como se verá más adelante.

Tras tener el sistema de vista funcionando pasaremos a crear un Template donde indicaremos como mostrar lo que deseamos, al ser Template Toolkit se puede utilizar algo similar a HTML<sup>2</sup> en el siguiente Cuadro 27.8 se puede ver el

<sup>1</sup>Template Toolkit Parte VI Página 81

<sup>2</sup>Manual Lenguaje HTML [http://es.wikibooks.org/wiki/Lenguaje\\_HTML](http://es.wikibooks.org/wiki/Lenguaje_HTML)

código generado que guardaremos en `root/src` .

### 27.3.2. Añadiendo el controlador

Ahora tenemos que crear el mecanismo necesario para poder ver nuestro mensaje y así mismo poder ver alguna respuesta por parte de este. Para ello tenemos que crear un Controlador, y de nuevo tenemos ayuda de Catalyst, mediante el uso del comando

```
script/ejemplo_create.pl controller hola
```

, nos creará un esqueleto que podremos manipular y que se puede ver en el Cuadro 27.9, esto nos crea un controlador inicial que no nos interesa, ya que lo mejor es modificar este para poder intercambiar mensajes entre la Vista y el Controlador, en el Cuadro 27.10 y el Cuadro 27.11 se puede ver las diferencias. Tras realizar este cambio tenemos un resultado que se puede ver en la Figura 27.3.

```
davioth@REN:~/.../Ejemplo$ script/ejemplo_create.pl controller hola
exists "/.../Ejemplo/script/..lib/Ejemplo/Controller"
exists "/.../Ejemplo/script/..t"
created "/.../Ejemplo/script/..lib/Ejemplo/Controller/hola.pm"
created "/.../Ejemplo/script/..t/controller_hola.t"
```

Cuadro 27.9: Creando el Controlador de Hola Mundo [18]

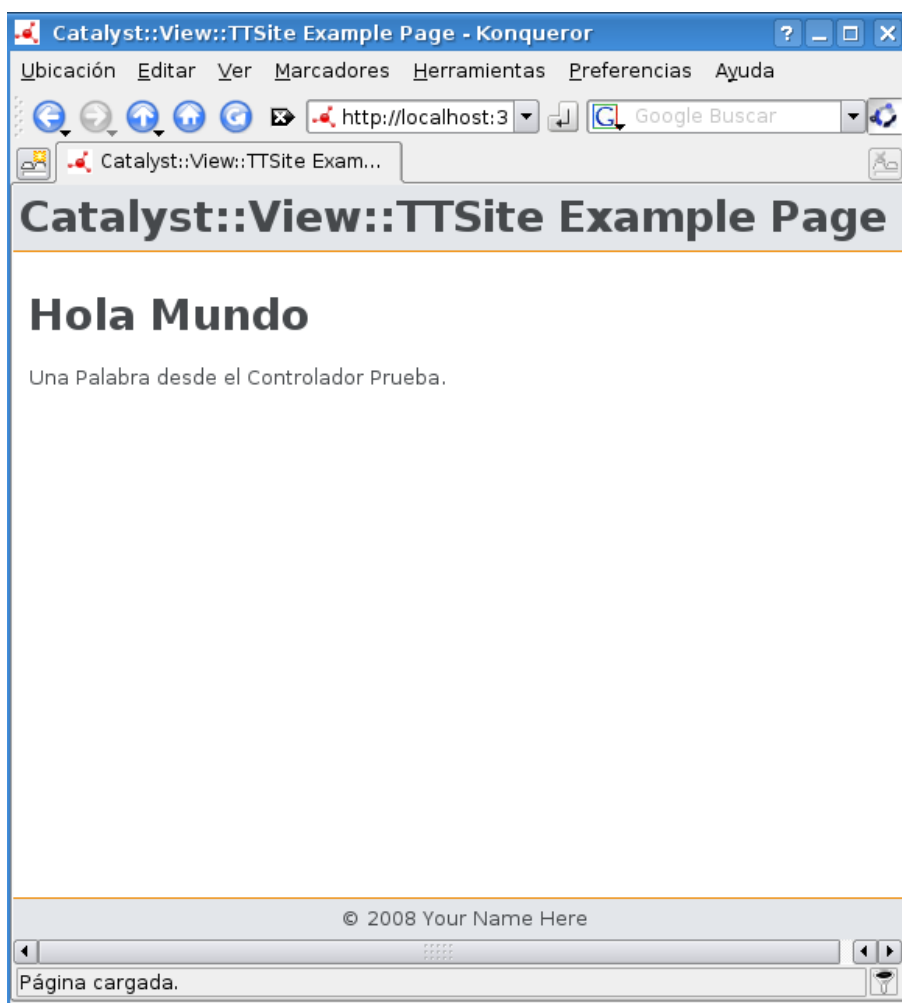


Figura 27.3: Hola Mundo en Catalyst

Lo que se ha hecho para poder pasar un mensaje al Template es sencillo,

### Controlador Original

```

1 package Ejemplo::Controller::hola;
2
3 use strict;
4 use warnings;
5 use base 'Catalyst::Controller';
6
7 =head1 NAME
8
9 Ejemplo::Controller::hola - Catalyst Controller
10
11 =head1 DESCRIPTION
12
13 Catalyst Controller.
14
15 =head1 METHODS
16
17 =cut
18
19
20 =head2 index
21
22 =cut
23
24 sub index : Private {
25     my ( $self, $c ) = @_;
26
27     $c->response->body('Matched Ejemplo::Controller::hola in hola.');
```

.....

```

31 =head1 AUTHOR
32
33 Ubay Díaz Machín,,
34
35 =head1 LICENSE
36
37 This library is free software, you can red...it and/or modify
38 it under the same terms as Perl itself.
39
40 =cut
41
42 1;
```

Cuadro 27.10: Creacion de la Vista 1 `/lib/Controller/hola.pm`

### Controlador Modificado

```

1 package Ejemplo::Controller::hola;
2
3 use strict;
4 use warnings;
5 use base 'Catalyst::Controller';
6
7 =head1 NAME
8
9 Ejemplo::Controller::hola - Catalyst Controller
10
11 =head1 DESCRIPTION
12
13 Catalyst Controller.
14
15 =head1 METHODS
16
17 =cut
18
19
20 =head2 index
21
22 =cut
23
24 sub hola : Global {
25     my ( $self, $c, @args ) = @_; #Se añade @args
    . para recoger argumentos
26 $c->stash->{template} = 'hola.tt2'; #Indicando el
    . template que lo Procesa
27 $c->stash->{word} = $word; #Se indica la variable en
    . el Template donde dirigir el mensaje
28
29 }
32 =head1 AUTHOR
35 Ubay Díaz Machín,,,
36
37 =head1 LICENSE
39 This library is free software, you can ... it and/or modify
40 it under the same terms as Perl itself.
42 =cut
44 1;

```

Cuadro 27.11: Creación de la Vista 2 `/lib/Controller/hola.pm`

### Creando la Vista para nuestro Ejemplo

```
davioth@REN:~/.../Ejemplo$ script/ejemplo_create.pl view TT TTSite
exists "/.../Ejemplo/script/.../lib/Ejemplo/View"
exists "/.../Ejemplo/script/.../t"
exists "/.../Ejemplo/script/.../lib/Ejemplo/View/TT.pm"
created "/.../Ejemplo/script/.../root/lib"
exists "/.../Ejemplo/script/.../root/src"
created "/.../Ejemplo/script/.../root/lib/config"
created "/.../Ejemplo/script/.../root/lib/config/main"
created "/.../Ejemplo/script/.../root/lib/config/col"
created "/.../Ejemplo/script/.../root/lib/config/url"
created "/.../Ejemplo/script/.../root/lib/site"
created "/.../Ejemplo/script/.../root/lib/site/wrapper"
created "/.../Ejemplo/script/.../root/lib/site/layout"
created "/.../Ejemplo/script/.../root/lib/site/html"
created "/.../Ejemplo/script/.../root/lib/site/header"
created "/.../Ejemplo/script/.../root/lib/site/footer"
exists "/.../Ejemplo/script/.../root/src/welcome.tt2"
exists "/.../Ejemplo/script/.../root/src/message.tt2"
exists "/.../Ejemplo/script/.../root/src/error.tt2"
exists "/.../Ejemplo/script/.../root/src/ttsite.css"
```

### Estructura de directorio creada

```
|-- lib
|  |-- config
|  |  |-- col
|  |  |-- main
|  |  '-- url
|  '-- site
|     |-- footer
|     |-- header
|     |-- html
|     |-- layout
|     '-- wrapper
|-- src
|  |-- error.tt2
|  |-- message.tt2
|  |-- ttsite.css
|  '-- welcome.tt2
```

Cuadro 27.7: Creación de la Vista

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns=" http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>Hola Mundo</title>
</head>
  <body>
<h1>Hola Mundo</h1>
<p>
Una Palabra desde el Controlador
[% word | html %].
</p>
  </body>
</html>
```

Cuadro 27.8: Creando el Template de Hola Mundo `/root/src/hola.tt2`

## Capítulo 28

### Expandiendo el Ejemplo

En este capítulo usaremos el esqueleto anterior para desarrollar nuestro ejemplo de Discoteca explicado en el Capítulo 27.2. Añadiremos la interface a la Base de Datos, y aprenderemos como crear Formularios.

#### 28.1. Añadiendo la Base de Datos

Catalyst usa para el acceso a la base de datos DBIx<sup>1</sup>, lo que nos permitirá un acceso más lógico a las bases de datos, ya que no nos fuerza a utilizar SQL<sup>2</sup>, pero con toda la potencia que nos puede proporcionar una Base de Datos.

Para nuestro ejemplo usaremos como sistema gestor de Bases de Datos SQLite<sup>3</sup>, por varias razones pero la principal es que es un sistema ligero y sencillo para el ejemplo planteado. De todos modos se debe tener en cuenta que se puede utilizar cualquier otro sistema de Base de Datos. Ahora veremos como crear la Base de Datos y configuraremos el sistema.

##### 28.1.1. Creando la Base de Datos

Lo primero que vamos a hacer es diseñar las secuencias SQL para crear nuestra Base de Datos. Como se puede ver en el Cuadro 28.1, tras esto insertaremos unos datos de prueba para las siguientes pruebas, como se puede ver en el Cuadro 28.2.

---

<sup>1</sup>Introducción a DBIx::Class Parte V Página 59

<sup>2</sup>Introducción a SQL Parte II Página 7

<sup>3</sup>SQLite Parte III Página 25

**Creando la Base de Datos**

```
davioth@kathia:~/.../Ejemplo$ sqlite3 datos.db
SQLite version 3.4.2
Enter ".help" for instructions
sqlite> CREATE TABLE info (
...> IDINFO INTEGER PRIMARY KEY AUTOINCREMENT,
...> DATOS TEXT NOT NULL
...> );
sqlite> CREATE TABLE localizacion (
...> IDLOCALIZACION INTEGER PRIMARY KEY AUTOINCREMENT,
...> DESCRIPCION TEXT NOT NULL
...> );
sqlite> CREATE TABLE tipo (
...> IDTIPO INTEGER PRIMARY KEY AUTOINCREMENT,
...> DESCRIPCION TEXT NOT NULL
...> );
sqlite> CREATE TABLE item (
...> ID INTEGER PRIMARY KEY AUTOINCREMENT,
...> TITULO TEXT NOT NULL,
...> LUGAR INTEGER NOT NULL ,
...> KIND INTEGER NOT NULL
...> );
sqlite> CREATE TABLE posee (
...> ID_INFO INTEGER,
...> ID_ITEM INTEGER,
...> PRIMARY KEY (ID_INFO, ID_ITEM)
...> );
```

Cuadro 28.1: Creación de la Base de Datos de Ejemplo

### Insertando en la Base de Datos

```

sqlite> INSERT INTO tipo VALUES (null, 'Animación');
sqlite> INSERT INTO tipo VALUES (null, 'Drama');
sqlite> INSERT INTO tipo VALUES (null, 'Acción');
sqlite> select * from tipo;
1|Animación
2|Drama
3|Acción
sqlite> INSERT INTO localizacion VALUES (null, 'Estanteria');
sqlite> INSERT INTO localizacion VALUES (null, 'Tarrina 1.1');
sqlite> select * from localizacion;
1|Estanteria
2|Tarrina 1.1
sqlite> INSERT INTO item VALUES (null, 'Saint Seiya',1,1);
sqlite> INSERT INTO item VALUES (null, 'Iron Man',3,2);
sqlite> select * from item
...> ;
1|Saint Seiya|1|1
2|Iron Man|3|2
sqlite> INSERT INTO info VALUES (null,'Película basada ... Marvel');
sqlite> INSERT INTO info VALUES (null,'Serie de Animación ... 90');
sqlite> select * from info;
1|Película basada en el Comic de Marvel
2|Serie de Animación de los Años 90
sqlite> INSERT INTO posee VALUES(1, 2);
sqlite> INSERT INTO posee VALUES(2, 1);
sqlite> select * from posee;
1|2
2|1
sqlite>

```

Cuadro 28.2: Inserción de datos Base de Datos de Ejemplo

#### 28.1.2. Creando el Esquema de la Base de Datos en Catalyst

Tras tener nuestra base de datos creada el siguiente paso es indicar el esquema de nuestra base de datos. Para ello lo primero es instalar lo necesario para lo que realizo lo siguiente

```
cpan -i Catalyst::Model::DBIC::Schema
```

para instalar DBIx para Catalyst. Cuando ya tengamos instalado el sistema deberemos configurar las relaciones de la Base de Datos pero esto se puede hacer de diferentes formas, en nuestro caso lo haré con el método estático donde seremos nosotros los que estableceremos como se puede ver en los siguientes Cuadros.

#### Indicando las Tablas de Nuestra Base de Datos

```
1 package EjemploDB;
2
3 use base qw/DBIx::Class::Schema/;
4
5 __PACKAGE__->load_classes({
6     EjemploDB => [qw/Item Info Posee Localizacion Tipo/]
7 });
8
9 1;
10
11
```

Cuadro 28.3: Clases que utilizaran nuestra Base de Datos situado en el Fichero `/lib/EjemploDB.pm`

En el Cuadro 28.3 se puede ver el fichero `Ejemplos/lib/Ejemplo.pm` donde estableceremos las tablas que nos permitirán manejar la base de datos.

A continuación tendremos que configurar el resto de nuestra tablas, para ello crearemos un directorio en **lib**, donde se editaran los esquemas del resto de tablas. Por facilidad se pueden ver primero aquellas tablas donde es muy simple como se puede ver en el Cuadro 28.4.

**Esquema de la Tabla Info**

```
1 package EjemploDB::Info;
2
3 use base qw/DBIx::Class/;
4
5 # Load required DBIC stuff
6 __PACKAGE__->load_components(qw/PK::Auto Core/);
7 # Se indica la tabla en la base de datos
8 __PACKAGE__->table('info');
9 # Se indica las columnas de la tabla
10 __PACKAGE__->add_columns(qw/IDINFO DATOS/);
11 # Se establece la Clave Priamria de la Tabla
12 __PACKAGE__->set_primary_key(qw/IDINFO/);
13
14 # Relación Posee
15 #
16 __PACKAGE__->has_many (info_item => 'EjemploDB::Posee', 'ID_INFO');
17
18 1;
```

Cuadro 28.4: Clases que representa la Tabla Info situado en el Fichero **/lib/EjemploDB/Info.pm**

En el Cuadro 28.4 se puede ver el fichero de configuración de la Tabla Info, donde se ha establecido los principales parámetros de esta. En ella se puede ver la relación con la tabla Posee.

### Esquema de la Tabla Tipo

```

1 package EjemploDB::Tipo;
2
3 use base qw/DBIx::Class/;
4
5 # Load required DBIC stuff
6 __PACKAGE__->load_components(qw/PK::Auto Core/);
7 # Nombre de la Tabla
8 __PACKAGE__->table('tipo');
9 # Columnas en la Tabla
10 __PACKAGE__->add_columns(qw/IDTIPO DESCRIPCION/);
11 # Se establece la clave principal
12 __PACKAGE__->set_primary_key(qw/IDTIPO/);
13
14 1;
```

Cuadro 28.5: Clases que representa la Tabla Tipo situado en el Fichero `/lib/EjemploDB/Tipo.pm`

Lo que se puede ver en el Cuadro 28.5 son los parámetros de la tabla Tipo.

### Esquema de la Tabla Localización

```

1 package EjemploDB::Localizacion;
2
3 use base qw/DBIx::Class/;
4
5 # Load required DBIC stuff
6 __PACKAGE__->load_components(qw/PK::Auto Core/);
7 # Nombre de la tabla
8 __PACKAGE__->table('localizacion');
9 # Conjunto de columnas
10 __PACKAGE__->add_columns(qw/IDLOCALIZACION DESCRIPCION/);
11 # Set the primary key for the table
12 __PACKAGE__->set_primary_key(qw/IDLOCALIZACION/);
13
14
15 1;
```

Cuadro 28.6: Clases que representa la Tabla Localización situado en el Fichero `/lib/EjemploDB/Localizacion.pm`

En el Cuadro 28.6 se pueden ver los parámetros de la tabla Localización.

Esquema de la Tabla Item	
1	package EjemploDB::Item;
2	
3	use base qw/DBIx::Class/;
4	
5	# Load required DBIC stuff
6	__PACKAGE__->load_components(qw/PK::Auto Core/);
7	# Set the table name
8	__PACKAGE__->table('item');
9	# Set columns in table
10	__PACKAGE__->add_columns(qw/ID TITULO/);
11	# Set the primary key for the table
12	__PACKAGE__->set_primary_key(qw/ID/);
13	
14	# Conjunto de claves ajenas
15	__PACKAGE__->has_many(lugar => 'EjemploDB::Localizacion',
16	'IDLOCALIZACION');
17	__PACKAGE__->has_many(kind => 'EjemploDB::Tipo', 'IDTIPO');
18	__PACKAGE__->has_many(info => 'EjemploDB::Info', 'IDINFO');
19	
20	#Para relación Posee
21	
22	__PACKAGE__->has_many (item_info => 'EjemploDB::Posee', 'ID_ITEM');
23	1;

Cuadro 28.7: Clases que representa la Tabla Item situado en el Fichero `/lib/EjemploDB/Item.pm`

En el Cuadro 28.8 se establecen los parámetros de la tabla Item que tiene varias relaciones del tipo 1 a Muchos que nos permitirán acceder a los datos de una forma directa, como por ejemplo **kind** nos permitirá acceder a la fila asociada en la tabla Tipo.

Esquema de la Tabla Posee
---------------------------

1 package EjemploDB::Posee;
2
3 use base qw/DBIx::Class/;
4
5 # Load required DBIC stuff
6 __PACKAGE__->load_components(qw/PK::Auto Core/);
7 # Set the table name
8 __PACKAGE__->table('posee');
9 # Set columns in table
10 __PACKAGE__->add_columns(qw/ID_INFO ID_ITEM/);
11 # Set the primary key for the table
12 __PACKAGE__->set_primary_key(qw/ID_INFO ID_ITEM/);
13
14 #Claves Ajenas
15
16 __PACKAGE__->belongs_to (item => 'EjemploDB::Item', 'ID_ITEM');
17 __PACKAGE__->belongs_to (info => 'EjemploDB::Info', 'ID_INFO');
18
19
20 1;

Cuadro 28.8: Clases que representa la Tabla Posee situado en el Fichero `/lib/EjemploDB/Posee.pm`

En el Cuadro 28.8 se puede ver la configuración de la Tabla Posee, que posee varias claves ajenas por lo que se utiliza **belongs\_to** para establecer estas relaciones. Tras configurar los parámetros de todas las tablas se debe crear el esquema para que el sistema pueda manejarla, para ello Catalyst nos proporciona lo necesario ya que sólo tendremos que ejecutar el comando del Cuadro 28.9. La forma es indicar que se quiere un modelo de la Base de Datos con **model**,

### Creando el Modelo

```
davioth@kathia:~/.../Ejemplo$ script/ejemplo_create.pl model
EjemploDB DBIC::Schema EjemploDB dbi:SQLite:datos.db
'' '' '{ AutoCommit => 1 }'
exists "/.../Ejemplo/script/../../lib/Ejemplo/Model"
exists "/.../Ejemplo/script/../../t"
created "/.../Ejemplo/script/../../lib/Ejemplo/Model/EjemploDB.pm"
created "/.../Ejemplo/script/../../t/model_EjemploDB.t"
```

Cuadro 28.9: Creando el Modelo

La estructura del Modelo de la Base de Datos se puede ver en el Cuadro 28.10.

### Modelo de la Base de Datos

```
|-- EjemploDB
| |-- Info.pm
| |-- Item.pm
| |-- Localizacion.pm
| |-- Posee.pm
| '-- Tipo.pm
```

Cuadro 28.10: Árbol de Directorios del Modelo

Como se puede ver en el Cuadro 28.11 se puede ver como al iniciar el servidor las nuevas Clases creadas.

Ejecución con las Nuevas Clases	
Class	Type
Ejemplo::Controller::Root	instance
Ejemplo::Controller::hola	instance
Ejemplo::Model::EjemploDB	instance
Ejemplo::Model::EjemploDB::Info	class
Ejemplo::Model::EjemploDB::Item	class
Ejemplo::Model::EjemploDB::Localizacion	class
Ejemplo::Model::EjemploDB::Posee	class
Ejemplo::Model::EjemploDB::Tipo	class
Ejemplo::View::TT	instance

Cuadro 28.11: Los Nuevos Modelos añadidos

Como nota se debe tener en cuenta que al crear la configuración se pueden establecer más parámetros, como el tipo de columna o tamaños pero que serán ampliados. Pero algo muy importante es que además de configurar las tablas de pueden añadir funciones que nos permitan realizar funciones sobre las tablas al estilo de procedimientos almacenados de las bases de datos.

## 28.2. Mostrando los Datos Almacenados

Lo que se quiere mostrar a partir de ahora es como realizar una web que nos muestre los datos de la Base de Datos que ya hemos almacenado para ilustrar este caso. Para que la forma de hacerlo sea más clara primero sólo buscaremos mostrar los datos almacenados en la tabla **Item** sin mostrar datos de las otras tablas.

### 28.2.1. Mostrando Item

Para poder mostrar los datos de la tabla **item** se debe añadir un controlador que maneje esto. Básicamente definiremos un método que realice una consulta a la base de datos y se lo pase al template creado para mostrar el contenido. Para ello lo primero es utilizar las herramientas incluidas para crear el esqueleto de nuestro Controlador, como se puede ver en la Sección 27.3.2 en la página 125,

pero en nuestro caso crearemos el controlador **Item** como se puede ver en el Cuadro 28.12.

#### Añadiendo el Controlador Item

```
davioth@kathia:~/.../Ejemplo$ script/ejemplo_create.pl controller Items
exists "/.../Ejemplo/script/../../lib/Ejemplo/Controller"
exists "/.../Ejemplo/script/../../t"
created "/.../Ejemplo/script/../../lib/Ejemplo/Controller/Items.pm"
created "/.../Ejemplo/script/../../t/controller_Items.t"
```

Cuadro 28.12: Añadiendo el Controlador Item

Tras haber creado el esqueleto de nuestro controlador como se vio en el Cuadro 27.11, en este caso se ha añadido una función llamada **list** que realiza una consulta a la base de datos obteniendo todas las entradas de la tabla **Item** como se puede ver en el Cuadro 28.13.

#### Listando el Contenido de Item

```
24 sub list : Local {
25     my ( $self, $c ) = @_;
26
27 #A continuación se selecciona la totalidad de elementos en la
28 #base de datos para ser transmitidos a a la variable items definida
29 #en template creado para Items
30     $c->stash->{items} = [$c->model('EjemploDB::Item')->all];
31
32 #Se indica el template que mostrará la información obtenida
33 #anteriormente, este estará en /root/src/item/list.tt2
34     $c->stash->{template} = 'item/list.tt2';
35 }
```

Cuadro 28.13: Función para el Listado de Contenido de Item situado en el Fichero **/lib/Controller/Item.pm**

En el Cuadro 28.13 se utiliza la la variable **items** del sistema para transmitir la información al template en la línea 30 y en la línea 34 se indica el Template que trata la información.

**Template Inicial para el Listado de Item**

```
1    [% META title = 'Lista de Items' -%]  
2  
3    <table>  
4    <tr><th>ID</th><th>Titulo</th></tr>  
5    [% # Muestra cada Item en Items%]  
6    [% FOREACH item IN items -%]  
7        <tr>  
8            <td>[% item.ID %]</td>  
9            <td>[% item.TITULO %]</td>  
10       </tr>  
11    [% END -%]  
12    </table>
```

Cuadro 28.14: Template Inicial para el Tratamiento de Item situado en el Fichero `/roo/src/Item/list.tt2`

En el Cuadro 28.14 se puede ver el Template que trata la información que el suministra el controlador mediante la variable **items** que se puede ver en la línea 6. Esta variable es un vector con las filas de la tabla, por lo que para recorrer esta tabla usamos un **FOREACH**, que ya fue explicado en la Sección 24.2.1 página 104, a continuación en la Figura 28.1 se puede ver el resultado.

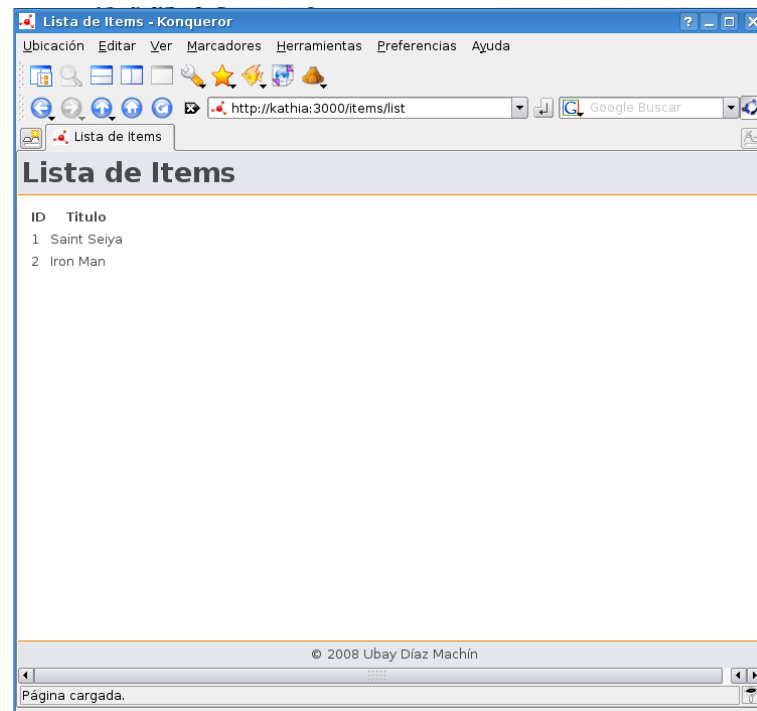


Figura 28.1: Web inicial con el Listado de Item

### 28.2.2. Mostrando Toda la Información

En esta sección abordaremos la forma de mostrar el contenido de la base de datos pero también mostrando los datos relacionados con cada fila de la tabla **Item**. Para lo único que se debe hacer es modificar el Template, añadiendo los datos que deseamos mostrar, sin tener que modificar el Controlador. Se ha modificado el fichero de configuración de la Tabla **Item** añadiendo una nueva relación. En el Cuadro 28.15 se puede ver esta nueva relación que permite acceder a la tabla **Info** directamente.

#### Nueva Relación en Item

```
__PACKAGE__->has_many (item_info => 'EjemploDB::Posee', 'ID_ITEM');
__PACKAGE__->many_to_many (infos => 'item_info', 'info');
```

Cuadro 28.15: Relación Añadida a la Tabla Item situado en el Fichero `/lib/EjemploDB/Item.pm`

Como se puede ver en el Cuadro 28.15 se crea una relación **infos** hacia la relación anterior **item\_infos** que relacionaba la columna **ID\_ITEM**, de esta manera

para acceder a la entra de la tabla **Info** sólo se debe usar la relación **infos**, como por ejemplo así

```
item.infos
```

A continuación se debe modificar el template anterior añadiendo los nuevos campos ha ser visualizados. como se puede ver en el Cuadro en donde se han utilizado las relaciones creadas anteriormente en la Sección 28.1.2. Como se ve en el Cuadro 28.16.

Mostrando más Datos	
1	<code>[% META title = 'Lista de Items' -%]</code>
2	
3	<code>&lt;table border = 1 &gt;</code>
4	<code>&lt;tr&gt;&lt;th&gt;ID&lt;/th&gt;&lt;th&gt;Titulo&lt;/th&gt;&lt;th&gt;Localización&lt;/th&gt;</code>
5	<code>&lt;th&gt;Tipo&lt;/th&gt;&lt;th&gt;Información&lt;/th&gt;&lt;/tr&gt;</code>
6	<code>[% # Muestra cada Item en Items%]</code>
7	<code>[% FOREACH item IN items -%]</code>
8	<code>&lt;tr&gt;</code>
9	<code>&lt;td&gt;[% item.ID %]&lt;/td&gt;</code>
10	<code>&lt;td&gt;[% item.TITULO %]&lt;/td&gt;</code>
11	<code>&lt;td&gt;[% item.lugar.DESCRIPCION %]&lt;/td&gt;</code>
12	<code>&lt;td&gt;[% item.kind.DESCRIPCION %]&lt;/td&gt;</code>
13	<code>&lt;td&gt;</code>
14	<code>[% tt_infos = [];</code>
15	<code>tt_infos.push (info.DATOS) FOREACH info = item.infos %]</code>
16	<code>([% tt_infos.size %])</code>
17	<code>[% tt_infos.join (', ') %]</code>
18	<code>&lt;/td&gt;</code>
19	<code>&lt;/tr&gt;</code>
20	<code>[% END -%]</code>
21	<code>&lt;/table&gt;</code>

Cuadro 28.16: Template Modificado para Mostrar más Datos de Item situado en el Fichero `/roo/src/list.tt2`

Como **item** es un objeto de la Base de Datos que nos permite acceder a la totalidad de sus campos incluido las relaciones. En la línea 15 se utiliza un **FOREACH** para acceder a la totalidad de informaciones en la tabla **Info**, Para poder seleccionar sólo los registros relacionados con la entrada actual, se relacionan únicamente los **info** que sean iguales a **infos** de la entrada actual de **item**.

ID	Título	Localización	Tipo	Información
1	Saint Seiya	Estanteria	Animación	(1) Serie de Animación ... 90
2	Iron Man	Tarrina 1.1	Drama	(1) Pelicula basada ... Marvel

Figura 28.2: Listado del Contenido de Item incluido información Adicional.

En la Figura 28.2 se puede ver el resultado.

### 28.3. Insertando Datos y Actualizando

En esta sección se mostrará como insertar nuevos objetos y actualizarlos, para ello se utiliza el añadido de Catalyst llamado `Catalyst::Controller::FromBuilder`<sup>4</sup>. Que nos permite crear formularios de una forma sencilla pero con un grado enorme de personalización. FormBuilder proporciona mecanismos de validación del contenido e indicación de campos obligatorios.

Para realizar esta operación lo primero puede ser configurar nuestro formulario que puede ser realizado de forma, una es la utilización de un fichero con extensión **fb** donde se establecen los campos como se puede ver en el Cuadro 28.17 otra forma es la de establecer el formulario en el programa que lo utiliza de forma dinámica como se ve en el Cuadro 28.18.

<sup>4</sup>Página Principal de FromBuilder <http://www.frombuilder.org>

Fichero edit.fb	
1	name: edit
2	method: post
3	fields:
4	title:
5	label: Título del Item
6	type: text
7	size: 40
8	required: 1
9	info:
10	label: Información Sobre el Medio
11	type: textarea
12	cols: 50
13	rows: 100
14	required: 1

Cuadro 28.17: Fichero de configuración del Formulario[23], situado en el Fichero `/root/forms/items/edit.fb`

Como se puede ver en el Cuadro 28.17 la forma de configurar los campos es sencilla ya que usa una forma de establecer similar a las de HTML<sup>5</sup>. En el Cuadro 28.18 se puede ver la forma dinámica de establecerlos que la posibilidad de tener campos cambiantes según selecciones anteriores.

---

<sup>5</sup>Formulario en Html [http://es.wikibooks.org/wiki/Lenguaje\\_HTML/\\_Formularios](http://es.wikibooks.org/wiki/Lenguaje_HTML/_Formularios)

### Formularios Dinámicos

```

61 #Campo para la selección de tipos de medios
62 $form->field(
63   name    => 'tipo',
64   type    => 'select',#Permite indicar el tipo
...           Select en el Formulario
65   options =>
66   [ map { [ $_->IDTIPO, $_->DESC... ] }
...         $c->model('EjemploDB::Tipo')->all],
67   label => 'Tipo de Medio',
68   required => 1
69 );

```

Cuadro 28.18: Estableciendo campos de formulario de forma dinámica situado en el Fichero `/lib/Controller/Item.pm`

Como se puede ver en el Cuadro 28.18 este campo es del tipo **select** que tiene como opciones el contenido de una tabla.

Tras tener nuestro formulario se debe establecer el Template que procesará este. En el Cuadro 28.19 se puede ver la forma más sencilla. FormBuilder también se pueden configurar permitiendo decidir cosas como la localización de los campos. Todo esto se puede ver en la página de FormBuilder. En el Cuadro 28.19 se puede ver el Template creado para ello.

### Template de Procesado de Formulario

```

1 [% META title = "Insertar y Editar Items" %]
2 [% FormBuilder.render %]

```

Cuadro 28.19: Template que Procesa el Formulario situado en el Fichero `/usr/src/item/edit.tt2`

Tras esto debemos añadir la función en nuestro Controlador que procese nuestra petición, para ello se ha cambiado el Template de listado que se puede ver en el Cuadro 28.16, añadiendo que el Título sea un enlace que es procesado por nuestra función donde se envía dos parámetros donde el primero es una bandera

para indicar que se está actualizando y el segundo es la clave principal del **Item**. En el caso actualizar lo primero es mostrar el contenido de la entrada y al enviar la información se realiza la actualización. En el Cuadro 28.20 se pueden ver la variaciones en el Template y en el Cuadro el código necesario.

Template de Mostrado de Información	
1	<code>[% META title = 'Lista de Items' -%]</code>
2	
3	<code>&lt;table border = 1 &gt;</code>
4	<code>&lt;tr&gt;&lt;th&gt;ID&lt;/th&gt;&lt;th&gt;Titulo&lt;/th&gt;</code>
5	<code>&lt;th&gt;Localización&lt;/th&gt;&lt;th&gt;Tipo&lt;/th&gt;&lt;th&gt;Información&lt;/th&gt;</code>
6	<code>&lt;th&gt;Eliminar&lt;/th&gt;&lt;/tr&gt;</code>
7	<code>[% # Muestra cada Item en Items%]</code>
8	<code>[% FOREACH item IN items -%]</code>
9	<code>&lt;tr&gt;</code>
10	<code>&lt;td&gt;[% item.ID %]&lt;/td&gt;</code>
11	<code>&lt;td&gt;&lt;a href="[% Catalyst.uri_for('edit/1/') _ item.ID %]"&gt;</code>
12	<code>[% item.TITULO %]&lt;/a&gt;&lt;/td&gt;</code>
13	<code>&lt;td&gt;[% item.lugares.DESCRIPCION %]&lt;/td&gt;</code>
14	<code>&lt;td&gt;[% item.kinds.DESCRIPCION %]&lt;/td&gt;</code>
15	<code>&lt;td&gt;</code>
16	<code>[% tt_infos = [];</code>
17	<code>tt_infos.push (info.DATOS) FOREACH info = item.infos %]</code>
18	<code>([% tt_infos.size %])</code>
19	<code>[% tt_infos.join (', ') %]</code>
20	<code>&lt;/td&gt;</code>
21	<code>&lt;td&gt;&lt;a href="[% Catalyst.uri_for('delete/') _ item.ID %]"&gt;</code>
22	<code>Eliminar&lt;/a&gt;&lt;/td&gt;</code>
23	<code>&lt;/tr&gt;</code>
24	<code>[% END -%]</code>
25	<code>&lt;/table&gt;</code>
26	<code>&lt;a href="[% Catalyst.uri_for('edit/0') %]"&gt;Añadir Elementos&lt;/a&gt;</code>

Cuadro 28.20: Template Variado en el Fichero `/root/src/Item/list.tt2`

La función **Catalyst.uri\_for** nos permite crear enlaces con la opción de envío de parámetros.

En el Cuadro 28.21 se puede ver código necesario para realizar la acción de actualización e inserción de registros.

Controlador de Inserción Actualización de Información
---

```

38 sub edit : Local Form {
39   my ($self, $c, $actu, $id) = @_;
40   my $id_info;
41   my $antkind;
42   .....
43
44
45   #Permite buscar un elemento para actualizar una entrada
46   my $item = $c->model('EjemploDB::Item')->find_or_new({ID => $id});
47   #Objeto para la creación de formularios
48   my $form = $self->formbuilder;
49   #Indicación del Template que renderiza el formulario
50   $c->stash->{template} = 'items/edit.tt2';
51
52   #Añadiendo el campo de selección de localización
53   $form->field(
54     name    => 'lugar',
55     type    => 'select',
56     options =>
57     [ map { [ $_->IDLOCALIZACION, $_->DESCRIPCION ] }
58       $c->model('EjemploDB::Localizacion')->all ],
59     label => 'Localización del Item',
60     required => 1 #Indica que el campo es obligatorio
61   );
62   #Campo para la selección de tipos de medios
63   $form->field(
64     name    => 'tipo',
65     type    => 'select',#Permite indicar el tipo Select
66     ... en el Formulario
67     options =>
68     [ map { [ $_->IDTIPO, $_->DESCRIPCION ] }
69       $c->model('EjemploDB::Tipo')->all ],
70     label => 'Tipo de Medio',
71     required => 1
72   );

```

Cuadro 28.21: Controlador Actualización Inserción de Información situado en el Fichero `/lib/Ejemplo/Controller/Item.pm`

En esta primera parte del código lo que se ilustra es el apartado de inicialización, como puntos a destacar está en el atributo **Form** añadido que indica el uso de **FormBuilder**, en el que también se puede indicar el fichero de configuración

de formulario. También se debe destacar en la línea 48 el momento en el que se instancia el objeto FormBuilder. Todo esto en el Cuadro 28.21.

### Controlador de Inserción Actualización de Información Continuación

```

72  if ($form->submitted && $form->validate) {
73
76  $infos = $c->model('EjemploDB::Info')->
...    find_or_new({IDINFO =>$id_info});
78  $item->TITULO($form->field('title'));
79
80      my $local = $form->field('lugar');
81      my $kind = $form->field('tipo');
82
83      $infos->DATOS($form->field('info'));
84      $infos->update_or_insert;
85      my $nitem = $item->update_or_insert;
86
87      if ($actu == 1) {
.....
90          #Se está actualizando se elimina primero los links
91          $item->delete_related ('item_kind', {ID_ITEM => $id});
92          $item->delete_related ('item_lugar', {ID_ITEM => $id});
93          $item->delete_related ('item_info', {ID_ITEM => $id});
94
95          $item->add_to_item_kind({ID_KIND => $kind});
96          $item->add_to_item_lugar({ID_LOCA => $local});
97          $item->add_to_item_info ({ID_INFO => $infos->IDINFO});
98      }
99      else {
100          #Se insertan nuevos valores
101          $item->add_to_item_info ({ID_INFO => $infos->IDINFO});
102          $item->add_to_item_kind({ID_KIND => $kind});
103          $item->add_to_item_lugar({ID_LOCA => $local});
104      }
.....
108          $c->forward('list');#Nos devuelve al listado
109      } else {

```

Cuadro 28.22: Controlador Actualización Inserción de Información Continuación en el Fichero `/lib/Ejemplo/Controller/Item.pm`

En el Cuadro 28.22 se muestra la parte del código que realiza la inserción y actualización de datos, en este cuadro destaca el uso de las funciones **add\_to\_item\_\*** que nos permite enlazar las tablas que tienen relaciones entre si.

#### Controlador de Inserción Actualización de Información Continuación

```

108 } else {
109   if ($id > 0) {
110
111     $id_info = $c->model('EjemploDB::Posee')->
... find({ID_ITEM => $id})->ID_INFO;
113     $infos = $c->model('EjemploDB::Info')->
.... find({IDINFO =>$id_info});
.....
117     $form->field(name => 'title', value => $item->TITULO);
118     $form->field(name => 'info', value => $infos->DATOS);
119   }
120
121 }
122 }

```

Cuadro 28.23: Controlador Actualización Inserción de Información parte Final en el Fichero **/lib/Ejemplo/Controller/Item.pm**

En este Cuadro 28.23 se puede ver la parte final del código que se encarga de cargar en los campos del formulario los datos ya almacenados para que no sirvan a la hora de la actualización.

El resultado final de los anteriores Cuadros se pueden ver en la Figura 28.3.

Figura 28.3: Ejemplo de Formulario para la Manipulación de Items

#### 28.4. Eliminada una Entrada

Para Eliminar una entrada se puede realizar de muchas formas, pero la forma elegida es la de pasar un identificador al controlador que elimina esta entrada. Para ello se debe modificar el Template como se ve en el Cuadro 28.24 añadiendo una nueva columna que es un enlace a la acción del controlador que elimina la entrada.

##### Eliminando Datos

```

17 </td>
18 <td><a href="[% Catalyst.uri_for('delete/') _ item.ID %]">
19 Eliminar</a></td>
20 </tr>

```

Cuadro 28.24: Template Modificado para permitir eliminar Datos de Item en el Fichero `/root/src/item/list.tt2`

Para añadir el enlace de eliminación como se puede ver en el Cuadro 28.24 se usa la función de Catalyst `uri_for`, que permite crear enlaces con parámetros,

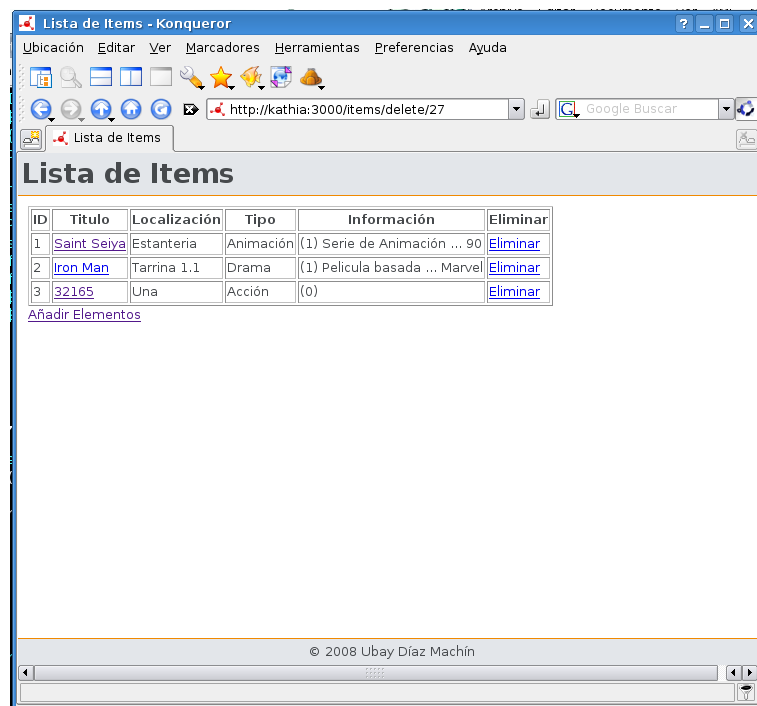


Figura 28.4: Muestra de Borrado de Item

donde el parámetro esta vez ha sido la primaria de **Item**. Para que se produzca la acción de borrado se utiliza el controlado que se puede ver en el Cuadro 28.25.

```

101 sub delete: Local {
102     my ($self, $c, $id) = @_;
103
104     #Permite eliminar una entrada de Item
105     $c->model('EjemploDB::Item')->find($id)->delete;
106
107     $c->forward('list');#Nos devuelve al listado
108 }

```

Cuadro 28.25: Controlador para la Eliminación de Items situado en el Fichero `/lib/Ejemplo/Controller/Item.pm`

En el Cuadro 28.25 en la línea 105 se realiza una búsqueda y a esta se le aplica la acción de borrado. Ya que el parámetro **\$id** es enviado de forma directa mediante la dirección web. Como se puede ver en la figura 28.4



## Capítulo 29

### Listado de Plugins

En este capítulo se puede ver un listado de alguno de los Plugins más interesantes disponibles para Catalyst, aunque es posible utilizar cualquier Módulo disponible en CPAN.

**Catalyst::Plugin::DateTime** Este plugin permite obtener Fechas y Tiempos desde Catalyst.

**Catalyst::Plugin::Wizard** Nos permite tener en nuestra aplicación. Asistente de una forma sencilla.

**Catalyst::Plugin::UploadProgress** Permite tener una barra que nos informa del Progreso de la subida de un fichero a un sitio Web.

**Catalyst::Plugin::Textile** Nos permite añadir un cuadro de texto que es transformado a HTML.

**Catalyst::Plugin::SuperForm** Permite crear Formularios Web de una forma sencilla.

**Catalyst::Plugin::StackTrace** Nos permite ver una traza de nuestra aplicación en la consola del servidor.

**Catalyst::Plugin::SimpleAuth** Nos permite tener un sistema de autenticación en nuestra aplicación de una forma sencilla.

**Catalyst::Plugin::Session** Nos permite gestionar la sesión de un usuario en nuestro sistema.

**Catalyst::Plugin::Server::XMLRPC** Permite tener funcionalidades de *XMLRPC* en nuestra aplicación.

**Catalyst::Plugin::Scheduler** Permite establecer eventos a realizar por nuestra aplicación, como por ejemplo enviar felicitaciones de cumpleaños a los usuarios.

**Catalyst::Plugin::Images** Permite obtener información sobre una imagen.

**Catalyst::Plugin::Geography** Nos proporciona información Geográfica.

**Catalyst::Plugin::FormValidator** Nos permite validar los campos de un Formulario.

**Catalyst::Plugin::FormCanary** Valida el número de campos enviados desde un formulario.

**Catalyst::Plugin::EmailValid** Permite validar un Email.

**Catalyst::Plugin::Email** Permite enviar emails desde Catalyst.

**Catalyst::Plugin::Continuation** Nos permite utilizar Continuation en Catalyst, que permite generar un flujo similar al de un programa.

**Catalyst::Plugin::CRUD** Permite tener Formularios fácilmente.

**Catalyst::Plugin::Browser** Permite detectar el navegador usado.

## Capítulo 30

### Notas Importantes

En este capítulo se quiere compartir algunas cosas aprendidas durante la redacción de este documento.

**Global** Si se pone a una función de un controlador la opción *Global*, este puede ser llamado directamente.

**Private** Permite declarar una función como privada de ese controlador, por eso sólo puede ser llamada cuando ya se está en la ruta de ese controlador.

**Controlador** Un controlador es traducido como parte de la dirección Web, como por ejemplo el controlador **test** es traducido como **server/test**.

**Depurar** Para depurar la aplicación la mejor forma que he visto es utilizar el propio depurador de Perl, para lo que llamaremos al depurador como se ve

```
perl -d script/parley_server.pl -k
```

y añadir en el código `$DB::single = 1;` para tener breakpoints. Tras esto se depura como otro código Perl, pero interactuando con el Navegador.

**Contenido Estático** Para acceder a este se debe ser colocada en **root/static** y dentro cualquier contenido será servido.

**Indicar Controlador de Inicio** La forma de establecer la acción por defecto es la de crear en el controlador **Root.pm** donde se modifica la función **sub default : Private**, donde estableceremos la acción por defecto, usando `$c->response->redirect($c->uri_for('/items'))`; y en el controlador **Item**, que es nuestro destino, tenemos que establecer una de la funciones como **sub index : Global** para que de esta manera sea la acción por defecto al direccionar.



**Parte VIII**

**Conclusiones**



## Capítulo 31

### Conclusiones

#### 31.1. Lo Aprendido por el Autor

Durante la redacción de este trabajo he adquirido una serie de conocimientos muy interesantes ya que aunque tenía conocimientos sobre las bases de datos nunca las había utilizado mediante un objeto que las representara. Algo que ahora me parece la forma más lógica de utilizar las bases de datos dentro de una aplicación. Así mismo antes de realizar este trabajo creía que el uso de Templates o cualquier otro sistema de plantillas era una tarea complicada, pero he visto que Template Toolkit es una forma sencilla y lógica de realizar las cosas.

Así mismo nunca había utilizado una arquitectura Modelo-Vista-controlador y la verdad es que me ha parecido una forma muy natural de hacer las cosas, principalmente porque permite olvidarse de las partes más estéticas y centrarse únicamente en la forma de tratar la información. Y cuando ya se tenga el sistema funcionando, poder pasar a modificar los detalles específicos de la aplicación para mostrarla de una forma agradable al usuario final.

#### 31.2. Reflexiones sobre los utilizados

A la hora de realizar este trabajo se ha utilizado una serie de herramientas y conceptos de los cuales voy a opinar, brevemente:

**Latex** Aunque en un inicio pensaba que era una herramienta complicada para redactar un trabajo ha resultado ser una forma muy eficaz de producir un documento elegante a la par de sencillo.

**Kile** Es un editor muy eficaz a la hora de utilizar Latex ya que permite no tener que recordar la totalidad de opciones de Latex, ya que este proporciona menús para poder hacer esto.

**SQL** El lenguaje SQL se ha usado para explicar algunos conceptos sobre las bases

de datos, pero lo único que veo extraño son algunas de sus sentencias que pueden llevar a confundirlas.

**SQLite** Este motor de bases de datos ha resultado ser una sorpresa agradable, porque aunque no tiene la potencia de otros sistemas, si incluye algunas de las principales características necesarias en una base de datos. Como punto en su contra está el de no contemplar la integridad referencial entre claves y tablas. Pero teniendo en cuenta que es una base de datos pensada para ser usada incrustada en otros programas cumple bastante bien su función.

**DBI** Este módulo Perl, que no conocía anteriormente, me ha parecido un gran acercamiento al uso de las bases de datos de una forma más humana y sencilla. También se debe nombrar la gran cantidad de documentación existente.

**DBIx** Este módulo me ha sorprendido ampliamente ya que es una forma muy buena de conseguir utilizar las bases de datos de una forma humana, ya que en este caso se tienen las típicas funciones necesarias a la hora de manipular una base de datos. Como contrapartida me he encontrado con una documentación, pobre que espero mejore con el tiempo ya que el módulo se merece una mejor documentación.

**Template Toolkit** Este sistema de templates me ha parecido extraordinario ya que me ha parecido muy intuitivo a la hora de utilizarlos, ya que utiliza conceptos de múltiples lenguajes de programación.

**Catalyst** Ha sido la herramienta que más me ha sorprendido ya que permite en poco tiempo tener una aplicación web muy completa. El gran número de plugins hacen que la programación web sea sencilla y agradable. Además posee una gran comunidad de desarrolladores donde encontrar mucha información, aunque muchas veces no se tendrá directamente.

**Parte IX**

**Bibliografía**



## Bibliografía

- [1] «Catalyst Planet».  
<http://planet.catalystframework.org/>
- [2] «Cpan».  
<http://search.cpan.org>
- [3] «Manual SQL».  
<http://www.monografias.com/trabajos11/manu/manu.shtml>
- [4] «Template Toolkit Web».  
<http://template-toolkit.org/>
- [5] «Web SQLite».  
<http://www.sqlite.org>
- [6] «KDESVn», 2008.  
<http://kdesvn.alwins-world.de/trac.fcgi>
- [7] «Latex Project». Web, 2008.  
<http://www.latex-project.org/>
- [8] «SVN Project», 2008.  
<http://subversion.tigris.org/>
- [9] «Wikipedia SQL», 2008.  
[http://es.wikipedia.org/wiki/Celda\\_activa](http://es.wikipedia.org/wiki/Celda_activa)
- [10] «Wikipedia SQL Inglesa», 2008.  
<http://en.wikipedia.org/wiki/Sql>
- [11] BEKMAN, STAS y CHOLET, ERIC: *Practical mod\_perl*, 2003. ISBN 0-596-00227-0.
- [12] BERLIN, ASH:. «DBIx Class ResultSource».  
<http://search.cpan.org/~ash/DBIx-Class-0.08007/lib/DBIx/Class/ResultSource.pm>
- [13] —: «DBIx Class Manual Cookbook». CPAN, 2008.  
<http://search.cpan.org/~ash/DBIx-Class-0.08007/lib/DBIx/Class/Manual/Cookbook.pod>
- [14] —: «DBIx Class Relationship». CPAN, 2008.  
<http://search.cpan.org/~ash/DBIx-Class-0.08007/lib/DBIx/Class/Relationship.pm>

- [15] —: «DBIx Class Relationship Base», 2008.  
<http://search.cpan.org/~ash/DBIx-Class-0.08007/lib/DBIx/Class/Relationship/Base.pm>
- [16] BUNCE, TIM: «Cpan DBI». CPAN, 2008.  
<http://search.cpan.org/~timb/DBI-1.604/DBI.pm>
- [17] CHAMBERLAIN, DARREN; CROSS, DAVID y WARDLEY, ANDY: *Perl Template Toolkit*. O'Reilly, 2003. ISBN 0-596-00476-1.  
<http://examples.oreilly.com/perl/tt/>
- [18] DIMENT, KIEREN: «Catalyst Manual Tutorial». CPAN, 2008.  
<http://search.cpan.org/author/ZARQUON/Catalyst-Manual-5.7012/lib/Catalyst/Manual/Tutorial.pod>
- [19] LARRY WALL, RANDAL SCHWARTZ: *Programming the Perl DBI: Database Programming with Perl*. Oreilly, 2000. ISBN 1565926994.
- [20] PECHTA, JONATHAN; ZENITH, FEDERICO; DANIELSSON, HOLGER; BRAUN, THOMAS y LUDWIG, MICHEL: «Kile Web», 2007.  
<http://kile.sourceforge.net/>
- [21] ROCKWAY, JONATHAN: *Catalyst Accelerating Perl Web Application Development*. Packt Publishing Ltd, 2007. ISBN 978-1-847190-95-6.
- [22] TO SQLITE, THE DEFINITIVE GUIDE: *The Definitive Guide to SQLite*. Apress, 2006. ISBN 9781590596739.
- [23] WIGER, NATE: «CGI FormBuilder». CPAN, 2008.  
<http://search.cpan.org/~nwiger/CGI-FormBuilder-3.0501/lib/CGI/FormBuilder.pod>
- [24] WIKIPEDIA: «Comparación de Frameworks Web». Web, 2008.  
[http://en.wikipedia.org/wiki/Comparison\\_of\\_web\\_application\\_frameworks](http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks)

**Parte X**

**Apéndices**



## Apéndice A

### GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

#### 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

#### 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The Cover Texts are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using

a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section ".<sup>En</sup>titled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as ".<sup>A</sup>cknowledgements", "Dedications", ".<sup>En</sup>dorsements", or "History".) To "Preserve the Title." of such a section when you modify the Document means that it remains a section ".<sup>En</sup>titled XYZ." according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

### 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

### 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

\* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

\* B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

\* C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

\* D. Preserve all the copyright notices of the Document.

\* E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

\* F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

\* G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

\* H. Include an unaltered copy of this License.

\* I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

\* J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

\* K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

\* L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

\* M. Delete any section Entitled “.Endorsements”. Such a section may not be included in the Modified Version.

\* N. Do not retitle any existing section to be Entitled “.Endorsements” to conflict in title with any Invariant Section.

\* O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “.Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty

Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in

electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License or any later version applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not

as a draft) by the Free Software Foundation.