Ismael Calvo Villalvilla

51689141E

alaricoi@gmail.com

TRABAJO 3 DE EVALUACIÓN CONTINUA

Generación Automática de Código

Contenido

1 Introducción 2

2 Desarrollo de la práctica 3

2.1 Recursos de programación utilizados 3

2.2 Descripción (especificación) detallada de la solución 3

2.3 Alcance y limitaciones de la solución propuesta 10

2.4 Descripción de los casos de Prueba 10

3 Ubicación de los programas fuente y explicación del resto de directorios/ficheros anexados 12

4 Manuales de uso/instalación 13

5 Conclusiones. 16

# Introducción

En el presente documento se va describir la solución aportada al trabajo 3 de la asignatura de Generación Automática de Código del máster universitario Investigación en Ingeniería de Software y Sistemas

## Enunciado del trabajo Propuesto:

*Se trata de generar un programa “CRUD” (altas baja consulta y modificación de un registro de cualquier tabla de una base de datos definida en un SGBD relacional).*

*El generador tendrá como entrada el nombre de una tabla.*

*El usuario indica el nombre de la tabla (que deberá existir, obviamente, en el SGBD correspondiente) y el programa generador implementará una clase con, al menos, los métodos correspondientes de alta, baja, consulta y modificación de dicha tabla.*

*Se valorará positivamente:*

*Que el generador se pueda utilizar contra SGBD’s diferentes*

*Que el generador obtenga la clase en más de un lenguaje de salida*

*El alumno deberá proponer algún mecanismo de “extensibilidad” que permita incorporar validaciones suplementarias (“custom code”) sin perder la posibilidad de regeneración; esto es, cuando se regenere el programa se incorporarán dichas validaciones de manera dinámica.*

## Consideraciones Generales de la Solución

**Se va a utilizar parte del trabajo 2** para la captura de los datos referentes a la conexión a los diferentes SGBD, es decir, **MySQL y SQLite**.

Al utiliza jdbc nativo, el acceso a los metadatos del esquema seleccionado y de sus tablas es genérico por lo que la ampliación a otros SGDB,s resultaría poco invasiva.

**La generación de código se realizará en base a plantillas,** los motivos de la elección son:

* **Flexibilidad** a la hora de generar distinto código en diferentes lenguajes. En nuestro caso he optado por generar PHP y JAVA. Aunque hay parte e código dedicado la mayoría se basa en tag de sustitución por lo que incorporar otro lenguaje no sería traumático, si bien es verdad que el mayor esfuerzo lo conllevaría crear las pantallas necesarias.
* **Personalización no intrusiva,** es decir, modificar una la salida de generación la mayoría de las veces conlleva solo modificar la plantilla, evitando el mantenimiento del código del generador.

Por el contrario, el desarrollo de este tipo de soluciones conlleva un gran esfuerzo inicial ya que hay que definir previamente las posibilidades de condigo generado y trasladarlo a una plantilla. Con otros tipos generadores basados en la especificación del lenguaje el desarrollo sería más rápido pero estaríamos constreñidos a los lenguajes y arquitecturas seleccionados en un principio, sin posibilidad de flexibilizar el proceso.

# Desarrollo de la práctica

## Recursos de programación utilizados

Al igual que en los trabajos anteriores, el lenguaje elegido para el desarrollo es java (jdk 1.8), por los mismos motivos de:

* Comodidad en la gestión de SGDB,
* facilidad a la hora de trabajar con cadenas de caracteres (búsqueda, reemplazo, concatenación).
* Facilidad en el tratamiento de ficheros.
* Lenguaje de programación ampliamente difundido.

Las herramientas utilizadas para la construcción del programa son:

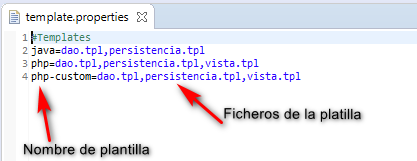
* Sistema operativo Windows 10.
* Como IDE (Integrated Development Environment) he utilizado Eclipse Oxigen en su instalación de Java.
* Librerías JAVA de terceros:
  + librería de SWT (Standard Widget Toolkit), de eclipse y openSource, nos permite crear de aplicaciones de ventanas y que crea la interfaz de usuario a partir de componentes operativos de cada Sistema operativo.
  + Librerías de conexión a SGDB. Para el trabajo hemos empleado SQLite y MySQL, sus conectores son los ficheros:
    - * mysql-connector-java-5.1.45-bin.jar
      * sqlite-jdbc-3.21.0.jar

## Descripción (especificación) detallada de la solución

Como se ha mencionado en la introducción, el generador se basa en plantillas, principalmente lo que se ha realizado es un trabajo previo de código bien formado en cada uno de los lenguajes y a partir de este código se han creado las diferentes plantillas.

Inicialmente se ha definido una arquitectura de ficheros de configuración donde podremos definir los lenguajes a generar y las plantillas de cada lenguaje, lo que nos permite ampliar el generador si modificar el su código fuente (o siendo lo menos invasivo posible).

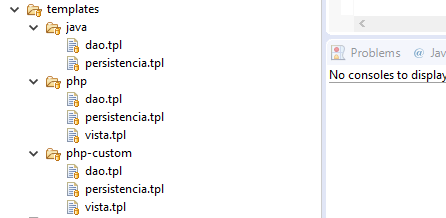
1. En la carpeta conf existe el fichero **template.properties** que nos servirá para definir el lenguaje y sus platillas.



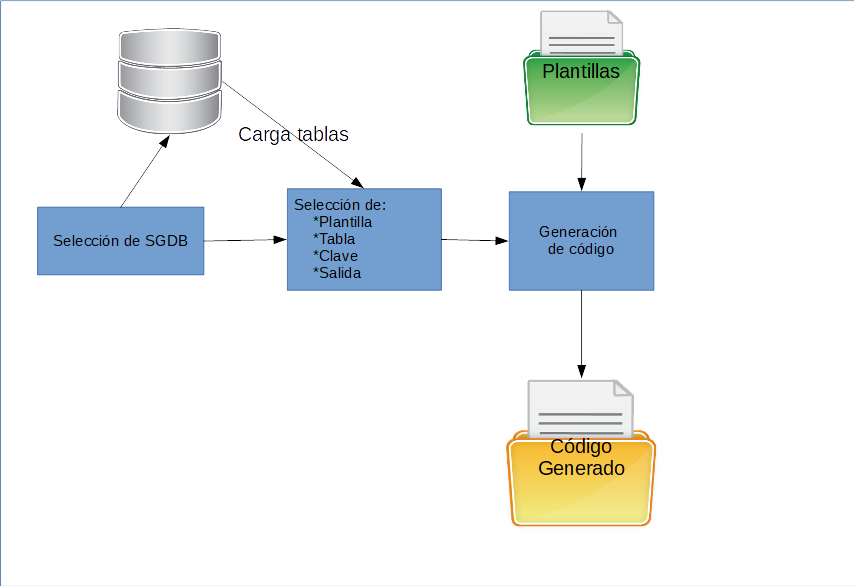
En el fichero de propiedades la clave es el nombre de plantilla y el valor cada uno de los ficheros de plantillas que luego generará un fichero de código fuente.

El nombre de plantilla está asociado al lenguaje, es decir, la extensión del fichero generado será el nombre la plantilla, como podemos tener varias configuraciones por lenguaje, he optado por que cada configuración debe empezar por su extensión de lenguaje hasta el carácter “–“, si no dispone de dicho carácter será todo el nombre la configuración

1. **Plantillas**. Las plantillas están situadas en la carpeta **templates**, cada configuración debe ser una carpeta y debe contener los ficheros definidos en **template.properties**



Una vez definidas las plantillas, el proceso de generación presenta las siguientes partes:



* Recogida de datos de conexión. El Generador recogerá los datos de conexión a base de datos y cargará los nombres de las tablas en un selector (comboBox).
* Recogida de los datos propios para la generación:
  + Plantilla: Se cargara un comboBox desde template.propeties con las platillas disponibles.
  + Tabla sobre la que queremos generar el CRUD. Desde el selector que se cargó en el punto anterior
  + Campo de la clave primaria de dicha tabla. Hay que tener en cuenta que muchos SGDBs son sensibles a mayúsculas y minúsculas.
  + Directorio de salida en filesystem.
* Generación propiamente dicha de los ficheros de código fuente a partir de las plantillas. Se han definido una serie de texto clave para su inclusión en la plantilla y que serán sustituidos por sus valores reales a la hora de generar el código, los tags definidos se presentan en la siguiente tabla.

| tag | | descripción |
| --- | --- | --- |
| <<#paquete\_modelos#>> | | En caso de java, el paquete donde reside la capa de persisitencia. En nuestro ejemplo será domain |
| <<#paquete\_daos#>> | | Java. Paquete de la capa de dao |
| <<#conexionBd#>> | | Cadena de conexón a los datos |
| <<#userDb#>> | | Usuario de conexión a base de datos |
| <<#nombre\_clave\_tabla#>> | | Nombre en base de datos de primaria de la tabla. Campo unico |
| <#nombre\_clave#>> | | Nombre de la vaiable del campo la clave en la generación de código |
| <<#nombreClase#>> | | Nombre que se asignara a la clase de persisitencia generada. |
| <<#nombreObjeto#>> | | La varible definida del tipo de la clase de persisitencia |
| <<#tablename#>> | | Nombre real de la tabla en SBDB |
| <<#todos\_campos\_tabla#>> | | Campos de la tabla en el SGDB separados por coma |
| <<#todos\_campos\_tabla\_parametro#>> | | Generación de campos de parámetro según el número de campos. |
| <<#todos\_campo\_valor#>> | | Cadena que genera el par clave valor a partir de los campos de la tabla. Se utiliza para el update de la tabla |
| <<#numero\_campos#>> | | Numero de campos de la tabla |
| "<<#seccion\_loop\_campos#>>  …..  <</#seccion\_loop\_campos#>> | | Etiqueta especial que nos permite realizar un proceso de bucle con tags relativos a campos. Se debe marcar con una etiqueta de cierre. Detro de este bloque van las siguientes entiquetas |
|  | <<#nombre\_campo#>> | Nombre de la variable campo con formato camelCase |
|  | <<#Nombre\_campo#>> | Igual que el anterior pero con la primera letra en mayúsculas. |
|  | <<#nombre\_campo\_tabla#>> | Nombre real del campo en la tabla del SGDB |
|  | <<#posicion\_campo#>> | Posición del campo en el sistema |
|  | <<#tipo\_campo#>>" | Tipo del campo trasformado al lenguaje, se emplea en las platillas Java. |

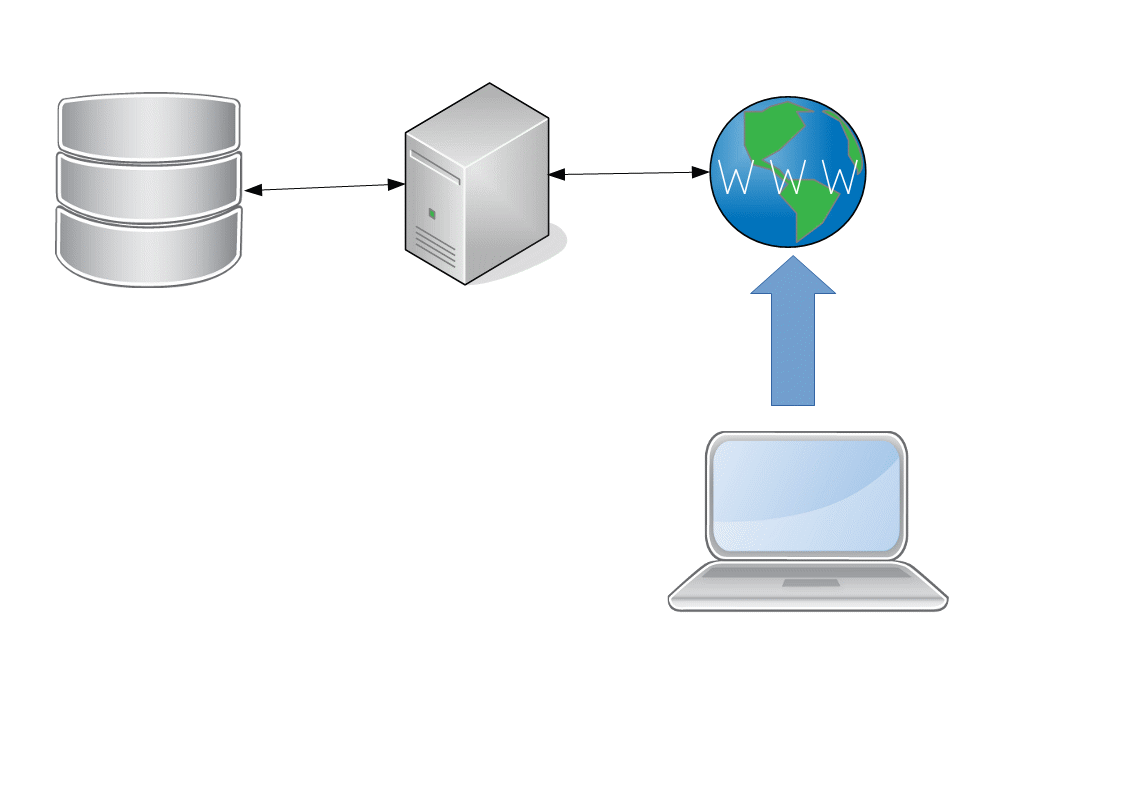
Para las pruebas de correcto funcionamiento del generador se ha partido de los siguientes casos:

### PHP

Se ha definido la salida del generador en tres ficheros dependiendo de su funcionalidad o capa funcional:

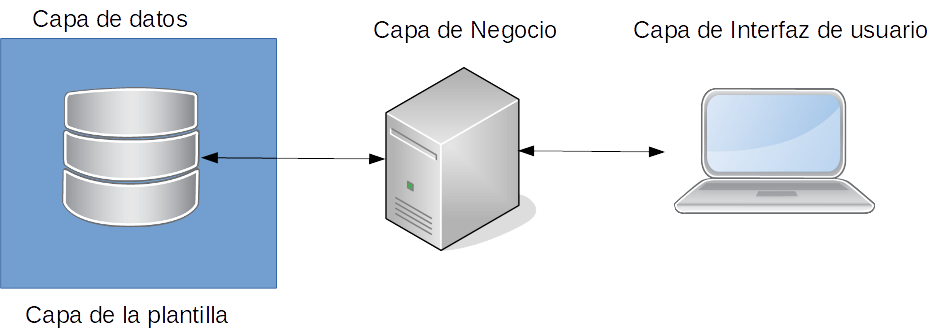
* Capa de Objeto de persistencia
* Capa de acceso a datos
* Capa de vista

Es un caso completo que nos generará una página funcional en el navegador, una clase dao que gestiona las acciones sobre los datos y una clase de persistencia que nos proporciona el objeto para trasladar las acciones desde la capa de dao a la vista.



### Java

Para este caso, me he centrado en la arquitectura de tres capas clásica, el generador solo creará la capa de acceso a datos y quedará pendiente del usuario del generador las llamadas de desde la capa de negocio. La elección de este tipo de generación es por la flexibilidad del código de generado ya que también se podrá utilizar en otras arquitecturas, como en desarrollo web la arquitectura Modelo-Vista-Controlador.



## Alcance y limitaciones de la solución propuesta

El generador solo contempla la generación de código sobre tablas con clave primaria de un solo campo, es decir, no contempla la posibilidad de tablas con claves primarias múltiples.

Las platillas para el caso de java se van a limitar a generar las clases de acceso a datos, es decir la capa de modelo en una arquitectura Model-Vista-Controaldor

## Descripción de los casos de Prueba

# Ubicación de los programas fuente y explicación del resto de directorios/ficheros anexados

# Manuales de uso/instalación

# Conclusiones.