(RE)GENERA Y ACTUALIZA TUS BASES DE DATOS CON

ALARIFE

VERSIÓN 1.0
29 DE AGOSTO DE 2009

INDICE

1. Introducción

1.1 ¿Qué es 'Alarife'?

Alarife es un componente software con licencia libre cuya función es facilitar el desarrollo del software permitiendo generar, regenerar y actualizar esquemas de bases de datos:

1. Re/genera el modelo de datos que utilice la aplicación, logrando que todos los desarrolladores y los testeadores de un mismo equipo manejen el mismo conjunto de datos.
2. Actualiza un modelo de datos que se encuentre en producción en caso de ser necesario.

1.2 ¿Para qué tipo de usuarios va destinada esta librería?

Dado su origen, Alarife está pensado para ser utilizado por desarrolladores y testeadores de software que basen su tecnología en Java y cuyos productos necesiten utilizar bases de datos.

También puede ser utilizada para facilitar la actualización de software en entornos de pre/producción.

1.3 Licencia

Actualmente, Alarife posee una licencia libre bastante permisiva denominada WTFPL (http://es.wikipedia.org/wiki/WTFPL)

2. ¿Por qué usar Alarife en nuestros desarrollos?

2.1 Escenarios de desarrollo y test

Cuando se desarrolla software que requiere del uso de una base de datos siempre se da uno de los dos los siguientes escenarios:

- Cada desarrollador posee **una base de datos propia** sobre la cual basa su desarrollo y con la que puede realizar algunas pruebas.
- El equipo de desarrollo posee una **base de datos centralizada** donde hay una carga de datos común a todos ellos.

Sin embargo, estos dos enfoques tienen inconvenientes que pueden entorpecer el desarrollo:

- Si cada desarrollador posee una base de datos distinta, cada uno poseerá cargas de datos distintas y además **centradas en apartados concretos de la aplicación**, con lo cual serán **incompletas**. El problema radica en que **no habrá un juego de datos completo** que pueda ser utilizado por todos y no es fácil que desarrolladores que están centrados en un módulo en concreto puedan contribuir a desarrollar otros.
- Si el equipo de desarrollo utiliza una base de datos centralizada se corre el riesgo de que **al eliminar o modificar registros se perjudique a otro desarrollador o a otra parte de la aplicación**, y no sólo eso, sino que quizá volver a generar datos que ya han sido borrados posiblemente sea una tarea **tediosa**.

Aparte estos dos enfoques dificultan el lanzamiento de tests automáticos que se realicen sobre el producto. Ya que es necesario disponer de una base de datos con la cual poder verificar que todo funciona correctamente, ¿invertimos

tiempo y esfuerzo en tener una base de datos aparte en la cual lanzar los tests?, ¿debería ser una copia de la base de datos de un desarrollador en concreto?, ¿de cuál?, ¿qué pasa si este desarrollador cambia los datos en base a su desarrollo o si son parciales?, si usamos una copia de la base de datos centralizada, ¿que pasa si los datos han cambiado con respecto a otros tests?

2.2 ¿Cómo unificar todo esto?

Alarife simplifica todo lo comentado en el apartado 2.1 enormemente, ya que con este componente la generación de bases de datos pasa a ser un proceso trivial. Gracias a ello, es bastante sencillo que:

- Todos los desarrolladores, aún con **bases de datos propias**, tengan la misma carga de datos. Esto **obliga** a que dichas cargas sean **completas**.
- Si se usa una base de datos centralizada y se borran o modifican datos, es **muy sencillo** volver a tener los mismos datos que se tenían **originalmente**.
- Puesto que las cargas de base de datos de desarrollo son completas, es **muy sencillo** crear un esquema que poder utilizar al lanzar los tests sin invertir **ningún esfuerzo** en ello.

Si se usa Alarife, además, se logra que los **datos de prueba útiles** para el desarrollo y para los tests pasen a **formar parte de la carga de datos** y los que no no se "enquistan" ya que no "da pena" borrarlos.

2.3 Ciclos de vida del desarrollo

En función a la fase dentro del ciclo de vida en la que se encuentre el proyecto

será deseable que Alarife realice unas acciones u otras:

- **Fase de desarrollo**: convendrá que Alarife genere y regenere los esquemas de la base de datos para tener la certeza de que durante esta fase todos los desarrolladores poseen la misma carga de base de datos.
- **Fase de test**: Alarife generará y regenerará los esquemas para poder basar las pruebas en el juego de datos que utilice el equipo de desarrollo.
- **Pase a pre/producción**: Alarife actualizará automáticamente el esquema de base de datos para que la nueva versión del software funcione correctamente.

3 Funcionamiento

3.1 ¿Cómo funciona Alarife?

El funcionamiento de Alarife es muy sencillo ya que se encarga de ir lanzado scripts que actualizan las tablas del modelo de datos y modifican la carga de las mismas.

De este modo, a medida que se producen cambios de versión en el software que necesiten cambios en el esquema de la base de datos se deben ir generando scripts que Alarife se encargará de ejecutar **nada más arrancar el programa**.

Pongamos un ejemplo. Tenemos un software que se encuentra en la versión 0.0.1, así que deberemos crear dos scripts, uno para crear las tablas y otro para actualizar el modelo de datos, de tal modo que la ejecución de ambos scripts dejen un esquema y una carga de datos acorde con la versión de la aplicación.

Más adelante en el desarrollo generamos la versión 0.0.2. Igualmente

deberemos crear dos scripts, uno que sirva para actualizar el modelo de datos y otro que añada más carga en la base de datos. Si tras la versión 0.0.2 se generase la versión 1.0.0 sería necesario generar otros dos ficheros y así sucesivamente.

Gracias a este mecanismo es muy sencillo que un desarrollador o un testeador tenga siempre una base de datos en estado consistente, ya que puede configurar Alarife para que al arrancar la aplicación éste se encargue de borrar la base de datos que ya existiese y fuese creando las tablas y los datos de las mismas de manera incremental.

En el ejemplo, cuando un desarrollador arranque la aplicación lanzará Alarife, el cual **borrará** la base de datos y ejecutará los scripts de la versión 0.0.1, tras ello los scripts de la versión 0.0.2, y finalmente los scripts de la versión 1.0.0.

De este modo **todos los datos que hayan introducido los desarrolladores para crear nuevas funcionalidades y hacer sus pruebas que no se encuentren en el script se perderán**. La necesidad de actualizar los scripts, por tanto, persigue tener una base de datos completa y con datos consistentes aunque haya varios desarrolladores trabajando en una misma aplicación pero en distintos módulos.

3.2 Sobre las actualizaciones

Como se ha comentado en el apartado 3.1, para regenerar la base de datos necesitamos que Alarife borrare la que tuviéramos anteriormente para generar una nueva automáticamente.

Sin embargo, como se indicó en el apartado 1.1 Alarife **también es capaz de actualizar bases de datos**.

Para lograr este comportamiento es necesario configurar Alarife e indicarle que debe actualizar la base de datos en vez de regenerarla y éste, llegado el

momento de la actualización, utilizará un mecanismo muy sencillo para poder determinar qué scripts debe ejecutar.

Volvamos al ejemplo del apartado 3.1. Si en producción está funcionando la versión 0.0.1 y es necesario actualizar a la versión 1.0.0, Alarife sabrá que debe lanzar los scripts de la versión 0.0.2 y de la 1.0.0 porque **en el modelo de datos debe existir una tabla llamada VERSION** donde se almacena la versión del software. De este modo Alarife recuperará una lista de scripts (los de las versiones 0.0.1, 0.0.2 y 1.0.0) y lanzará aquellos que sean mayores a los del número de versión que aparecen en la base de datos ordenados de menor a mayor (en este caso primero el de la versión 0.0.2 y después el de la versión 1.0.0).

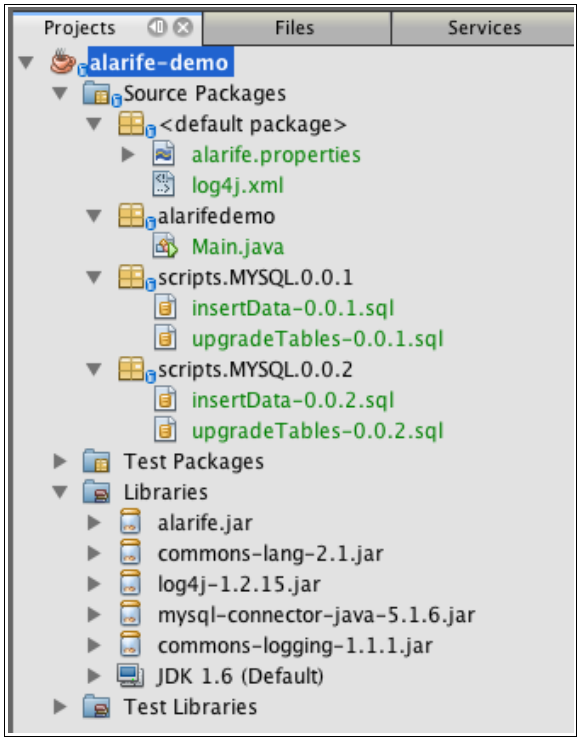
Gracias a este mecanismo se simplifica enormemente la actualización de modelos de datos en entornos de pre/producción.

4 Ejemplo de Funcionamiento

4.1 Ejemplo sencillo

Junto con la librería se puede descargar una aplicación de ejemplo muy pequeña que servirá para ver el uso más sencillo de Alarife con MySQL. Dicha aplicación se entrega de dos maneras, la primera de ellas es como proyecto de NetBeans y la segunda como proyecto de Eclipse.

Dicho ejemplo contiene los ficheros que se muestran en la siguiente figura:



Donde:

- 'alarife.properties' contiene los datos de conexión con la base de datos.
- 'log4j.xml' contiene la configuración de las trazas, las cuales tienen el umbral configurado a nivel INFO.
- 'alarifedemo.Main.java' contiene el programa que usa Alarife.
- 'Scripts.MYSQL.*' contiene los scripts que van actualizando la base de datos.

Vamos a centrarnos en los ficheros principales. El contenido de 'alarife.properties' es el siguiente:

```
# ------------------------------------------------------------------------ #
# Configuration of the database connection                                 #
#                                                                          #
# IMPORTANT! the instance MUST be finished by '/' and be careful with the  #
# lower and uppercase in 'schema', 'username' and 'password'               #
# ------------------------------------------------------------------------ #

instance = jdbc:mysql://localhost:3306/
```

```
driverClassName = com.mysql.jdbc.Driver
username = root
password = miContraseñaSuperSecreta

# ------------------------------------------------------------------------ #
# creation and destruction of the database once connected                  #
#                                                                          #
# IMPORTANT! this commands are database dependant, so it MUST NOT BE hardcoded #
# IMPORTANT! the database name MUST BE uppercase (for instance: MYSQL)     #
# ------------------------------------------------------------------------ #

# MySQL
MYSQL.dropDatabase = drop database if exists `futbol`;
MYSQL.createDatabase = create database if not exists `futbol`;
MYSQL.createVersionTable = create table `futbol`.`VERSION` (`version`
VARCHAR(10) NOT NULL);
MYSQL.insertVersionTable = insert into `futbol`.`VERSION` (`version`) values
('{0}');
MYSQL.recoverVersionTable = select `version` FROM `futbol`.`VERSION`;
```

Donde:

- 'instance' indica la conexión con el sistema gestor de bases de datos, en este caso MySQL.
- 'driverClassName' es el nombre del driver JDBC
- 'username' y 'password' son el nombre de un usuario con permisos para crear esquemas sobre la base de datos y su contraseña.

El resto de campos son dependientes de la base de datos e indican los comandos que deben usarse para:

- Borrar el esquema de la base de datos (en este ejemplo se llamará 'futbol').
- Crear el esquema de la base de datos.
- Crear la tabla con la versión (mirar apartado 3.2), crear su valor y recuperarlo.

Otros ficheros importantes son los scripts. Veamos el contenido de dos de ellos. Empecemos por 'upgradeTables-0.0.1.sql':

RAULEXPOSITO.com

```
CREATE TABLE `futbol`.`PLAYER` (
  `id` INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(255) NOT NULL,
  PRIMARY KEY(`id`)
) engine=innodb default charset=utf8 collate=utf8_spanish_ci;
```

Como se puede ver es un script con la creación de una tabla para MySQL. Alarife ejecuta primero estos ficheros con creación/edición/modificación de tablas y luego procede a ejecutar el script que modifica la carga, en este caso 'insertData-0.0.1.sql':

```
insert into `futbol`.`PLAYER` (`name`) values ('Casquero');
insert into `futbol`.`PLAYER` (`name`) values ('Belenguer');
insert into `futbol`.`PLAYER` (`name`) values ('Manu del Moral');
update `futbol`.`VERSION` set version = '0.0.1';
```

Es importante destacar que en este script se debe actualizar la versión. Como se han lanzado los scripts de la versión 0.0.1, **se debe actualizar la tabla 'VERSION' para reflejar que se han lanzado los scripts de esta versión**.

Finalmente, el contenido del programa que ejecuta Alarife es el siguiente:

```
package alarifedemo;

import com.raulexposito.alarife.exception.DatabaseException;
import com.raulexposito.alarife.manager.DatabaseManager;
import com.raulexposito.alarife.manager.enumeration.ApplicationMode;
import com.raulexposito.alarife.manager.enumeration.DatabaseType;
import com.raulexposito.alarife.manager.impl.DefaultDatabaseManager;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class Main {
```

```
    private static final Log log = LogFactory.getLog( Main.class );

                    private     static     final     String     DEVELOPMENT     =
ApplicationMode.DEVELOPMENT.toString();

    private static final String MYSQL = DatabaseType.MYSQL.toString();

    private static final String CONFIGURATION_FILE = "alarife.properties";

    public static void main(String[] args) {

        // declaracion del manager de Alarife
        final DatabaseManager databaseManager = new DefaultDatabaseManager();

         // indicamos que estamos en fase de desarrollo, de tal modo que borrara
la base de datos
        databaseManager.setApplicationMode(DEVELOPMENT);

        // indicamos que vamos a usar una base de datos MySQL
        databaseManager.setDatabaseType(MYSQL);

        // indicamos el fichero con la conexion con la base de datos
        databaseManager.setDatabasePropertiesFilename(CONFIGURATION_FILE);

        // se lanza Alarife para generar la base de datos
        try {
            databaseManager.execute();
        } catch (DatabaseException ex) {
            log.error ("Se produjo un error al generar la base de datos", ex);
        }
    }
}
```


RAULEXPOSITO.com

Como se puede apreciar es un código muy sencillo y se encuentra bien comentado. Los pasos que sigue son los siguientes:

1. Configura la lógica de Alarife para indicarle que va a ejecutarse en modo de desarrollo, con lo cual borrará el esquema de la base de datos y la regenerará por completo, y también le indica que la base de datos que va a utilizar es de tipo MySQL (mirar apartados 3.1 y 3.2).
2. Lee el contenido del fichero de propiedades 'alarife.properties', comentado anteriormente, de donde obtiene los datos de conexión con la base de datos.
3. Ejecuta su lógica, ejecutando los scripts de la versión 0.0.1 y después los de la versión 0.0.2

Al ejecutarse genera la siguiente traza de log:

```
INFO  DefaultDatabaseManager - the environment is configured as 'DEVELOPMENT'
with a database 'MYSQL'
INFO  SQLExecutor - database driver loaded: 'com.mysql.jdbc.Driver'
INFO  SQLExecutor - schema 'jdbc:mysql://localhost:3306/' dropped
INFO  SQLExecutor - schema 'jdbc:mysql://localhost:3306/' created
INFO  SQLExecutor - table 'VERSION' created
INFO  SQLExecutor - inserted the version '0.0.0' in the table 'VERSION'
INFO  SQLExecutor - the current version '0.0.0' is older than the latest version
'0.0.2', so an upgrade is needed
INFO  SQLExecutor - migrating to '0.0.1' version
INFO  SQLExecutor - reading the content of the upgrade tables script
INFO  SQLExecutor - reading the content of the insert data script
INFO  SQLExecutor - scripts succesfully executed [170 ms]
INFO  SQLExecutor - the current version '0.0.1' is older than the latest version
'0.0.2', so an upgrade is needed
INFO  SQLExecutor - migrating to '0.0.2' version
INFO  SQLExecutor - reading the content of the upgrade tables script
INFO  SQLExecutor - reading the content of the insert data script
INFO  SQLExecutor - scripts succesfully executed [134 ms]
```

Como se puede observar, borra el esquema de la base de datos, crea uno nuevo y va lanzando los scripts de manera incremental. De este modo al ejecutarse el programa podremos disponer de una base de datos con una

carga que pueda sernos de utilidad en nuestros desarrollos o en nuestros test, tal y como se indicó en los apartados 2.1, 2.2 y 2.3.

5 Contacto

Como no podía ser de otro modo, cualquier duda, sugerencia o contribución es bien recibida, o incluso si simplemente usas la librería para cualquier cosa te agradecería que me lo hicieras saber :-). Si quieres, puedes dejar un comentario en la entrada del proyecto, a la cual puedes acceder desde esta dirección:

http://raulexposito.com/projects.php

O bien escribirme a la siguiente dirección de correo electrónico:

raul@raulexposito.com

Espero que Alarife te haya podido ser de utilidad.