Persistencia con RedBeanPHP

Realizado por A.Garay (dpto.de Informática)

Resumen de contenidos

- Descarga e instalación
- Configuración de la BD
- Persistir un Bean
- Recuperar Beans
- Composición (m2o, o2m, m2m)
- Integración RedBeans + CodeIgniter

Descarga e instalación

- Descargar el fichero **rb.php** desde aquí
- Copiarlo a una carpeta dentro de nuestro proyecto (p.ej. lib)
- Para utilizar RedBean en cualquier script php, basta con ejecutar previamente un...
  - require_once('/ruta/a/**rb.php**')
- Cuidado con hacer varios includes (utilizar require_once)

Configuración de la BD (MySql)

- No hace falta configurar la cadena de conexión en ningún fichero externo, basta con ejecutar el comando, antes de ejecutar cualquier comando “RedBean”

```
R::setup('mysql:host=localhost;dbname=mydatabase', 'user', 'password')
```

- Para cerrar la conexión

```
R::close()
```

- Para otras bases de datos distintas a MySql, mirar aquí

Persistir un Bean

- A diferencia de Hibernate, la estructura (clases) de los Beans no hay que definirla previamente, basta crear un bean (con “dispense”, que es como un “new” para beans persistentes) y se le va dando estructura “por el camino”.
- **No se deben utilizar nombres de atributos que contengan <u>mayúsculas</u>**
  - `$libro= R::dispense( 'libro' );` // Nombre de la clase del bean en minúscula siempre
  - `$libro -> titulo = 'Aprender a programar';` // a través del atributo y no del setter
  - `$libro['precio'] = 29.99;` // formato array asociativo también funciona 😳
  - `$id = R::store( $libro);` // devuelve el lastInsertId por si hace falta

Recuperar/actualizar un Bean por id.

- Recuperar un Bean, conociendo su $id
  - $libro = R::load('libro', $id);
- Actualizar un Bean recuperado
  - Se podría modificar alguno de sus atributos.
    - $libro -> titulo = 'Otro título';
  - O “inventarse” atributos nuevos.
    - $libro -> genero = 'Aventuras';
  - Para luego persistirlo otra vez.
    - R::store($libro);

Borrar un Bean

- R::trash($libro);

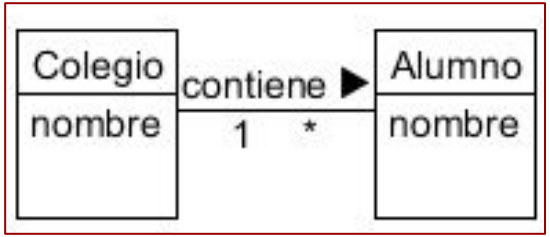
Consultas. Listas de beans

- Recuperar todos los beans de un tipo
  - $libros = R::**findAll**('libro');
- Recuperar beans por criterio WHERE de SQL
  - $libros = R::**find**('libro', ' precio > 10 ');
- Recuperar beans por criterio WHERE con parámetros anónimos
  - $libros = R::**find**('libro', ' titulo LIKE ? ', ['Aventura en%']);
- Recuperar beans por criterio WHERE con parámetros con nombre
  - $libros = R::**find**('libro', ' titulo LIKE :tit ', [':tit' => 'Aventura en%']);
- Realizar una consulta SQL genérica (devuelve array de array asoc.según SELECT)
  - $datos = R::**getAll** (“select c1,c2 from t1,t2 where t1.c4 = ‘algo’ “);
  - Ver también: **exec**, **getCol**, **getRow**. **getCell** y **getAssoc**

- Más información aquí

Composición: many to one

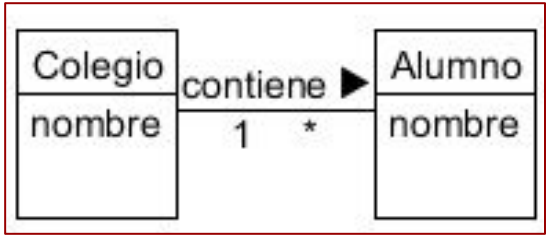
```
$colegio = R::dispense('colegio');
$colegio -> nombre = 'Rey Fernando VI';
$alumno = R::dispense('alumno');
$alumno -> nombre = 'Pepe';
$alumno -> colegio = $colegio;
R::store( $alumno );
```



- Más información aquí

Composición: one to many

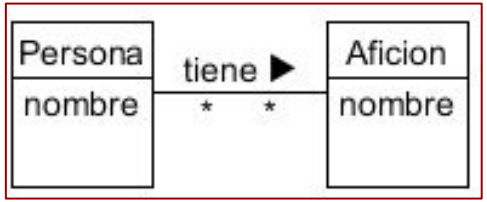
```
$colegio = R::dispense('colegio');
$colegio -> nombre = 'Rey Fernando VI';
$alumno = R::dispense('alumno');
$alumno -> nombre = 'Pepe';
$colegio -> ownAlumnoList [ ] = $alumno;
R::store( $colegio );
```



- Utilizar xown**Bean**List **[ ]** si se quiere borrado en cascada sobre los BEANs dependientes
- Más información aquí

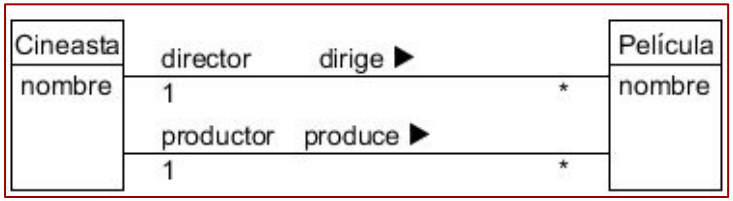
Composición: many to many

```
$pepe = R::dispense( 'persona' );$pepe->nombre='Pepe';
$futbol = R::dispense('aficion');$futbol->nombre='futbol';

$futbol -> sharedPersonaList [ ] = $pepe;
R::store( $futbol );

// ALTERNATIVAMENTE
// $pepe->sharedAficionList [ ] = $futbol;
// R::store( $pepe );
```



- Más información aquí

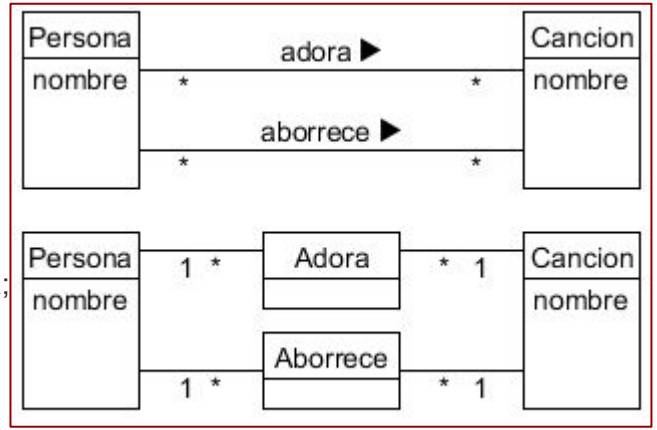
Aliasing en relaciones m2o ó o2m




```
$indiana = R::dispense('pelicula');$indiana -> nombre='Indiana Jones';
$starwars = R::dispense('pelicula');$starwars -> nombre='Star Wars';
$et = R::dispense('pelicula');$et -> nombre='E.T. el extraterrestre';

$lucas = R::dispense('cineasta'); $lucas -> nombre = 'George Lucas';
$kurtz = R::dispense('cineasta'); $kurtz -> nombre = 'Gary Kurtz';
$spielberg  = R::dispense('cineasta'); $spielberg -> nombre = 'Steven Spielberg';

// Ej. Construcción y almacenamiento desde el lado MANY TO ONE
$indiana -> director=$spielberg; $indiana -> productor = $lucas;
$starwars -> director=$lucas; $starwars -> productor = $kurtz;
$et -> director=$spielberg; $et ->productor = $spielberg;

R::storeAll( [$starwars, $indiana, $et] );

// Ej. Consultas desde el lado “ONE” (devolvería 'Indiana Jones, E.T. el extraterrestre')
$dirigidasPorSpielberg = $spielberg -> alias( 'director' ) -> ownPeliculaList;
foreach ($dirigidasPorSpielberg as $peli) {echo $peli -> nombre . ',';}
```

Aliasing en relaciones m2m

```
$alberto = R::dispense ( 'persona' ); $alberto->nombre = 'Alberto';
$pepe = R::dispense ( 'persona' ); $pepe->nombre = 'Pepe';

$paquito = R::dispense ( 'cancion' ); $paquito->nombre = 'Paquito el chocolatero';
$copacabana = R::dispense ( 'cancion' ); $copacabana->nombre = 'Copacabana';
$yesterday = R::dispense ( 'cancion' ); $yesterday->nombre = 'Yesterday';

$a_c = R::dispense('adora');$a_c->persona=$alberto;$a_c->cancion=$copacabana;
$a_y = R::dispense('adora');$a_y->persona=$alberto;$a_y->cancion=$yesterday;
$a_p = R::dispense('aborrece');$a_p->persona=$alberto;$a_p->cancion=$paquito;
$p_p = R::dispense('aborrece');$p_p->persona=$pepe;$p_p->cancion=$paquito;
$p_y = R::dispense('aborrece');$p_y->persona=$pepe;$p_y->cancion=$yesterday;
$p_c = R::dispense('aborrece');$p_c->persona=$pepe;$p_c->cancion=$copacabana;

R::storeAll( [$a_c, $a_y, $a_p, $p_p, $p_y, $p_c] ); // Persistir desde el lado “m2o”

$ama = $alberto -> aggr ( 'ownAdoraList', 'cancion' );  // Copacabana, Yesterday
echo 'Alberto ama...'; foreach ( $ama as $cancion ) { echo $cancion -> nombre . ' '; }

$aborrecidaPor = $paquito->aggr( 'ownAborreceList' , 'persona' ); // Pepe, Alberto
echo 'Paquito es aborrecida por...'; foreach ( $aborrecidaPor as $persona) {echo $persona -> nombre . ' ';}
```



¡¡ No se permite el aliasing en colecciones shared !!
SOLUCIÓN: convertir cada relación m2m, en dos relaciones o2m y m2o complementarias

RedBeanPHP y CodeIgniter

- Copiar **rb.php** en /application/third_party/rb
- Crear un fichero /application/libraries/**rb.php** que contenga lo siguiente:

```
class Rb {      function __construct()  {
                include(APPPATH.'/config/database.php');
                include(APPPATH.'/third_party/rb/rb.php');
                $host = $db[$active_group]['hostname'];
                $user = $db[$active_group]['username'];
                $pass = $db[$active_group]['password'];
                $db   = $db[$active_group]['database'];
                R::setup("mysql:host=$host;dbname=$db", $user, $pass); } }
```

- Modificar en /application/config/autoload.php la línea `$autoload['libraries'] = ['rb'];`
- Crear un modelo compatible con FUSE
  - Crear un modelo (bajo “model”), llamado Model_<Bean>, hijo de “RedBean_SimpleModel”
  - Añadir (sobrescribir) una función “public function update() {...}” para añadir reglas de negocio antes de persistir.
  - Añadir el resto de funciones DAO o FUSE (after_update(), dispense(), etc.) que consideremos oportunas.
  - Cargarlo desde el controlador como cualquier otro modelo “$this->load->model(‘modelo’,’aliasModelo’);

Resumen de comandos de RedBeanPHP

- https://docs.google.com/document/d/1qm_2bCtUBLLFKOvtRC5kc7WSi8j5-rVzvyn96EhRuT0/edit#heading=h.rb5qm6nihpep