RedBean ORM for PHP

The Idea

RedBean is a lightweight object relational mapping tool. The central concept of RedBean is the bean. A Bean is a simple object that acts as a data container. RedBean has two modes of operation: fluid and frozen. By default RedBean operates in fluid mode. This means that you can just store a bean using RedBean and it will adapt the database schemas if needed. If you are done developing you can freeze() RedBean and deploy to production environments.This is the idea: In fluid mode (default), **if you throw a bean at RedBean it will store that bean for you**, no matter what. That sounds easy doesnt it?

Getting Started

To start developing with RedBean, the easiest way is to use the kickstart method. A Kickstarter helps you to get started quickly without any prior initialization.

```
$toolbox = RedBean_Setup::kickstartDev(
"mysql:host=localhost;dbname=oodb","root","" );
```

The KickstartDev is meant for the development phase. It accepts three arguments; a database connection string (DSN), a username to access the database and finally a password (may be an empty string like in the example). Make sure you catch the output returned by this function; it consists of a very handy toolbox that you can use along the way. First make sure you get an instance of the RedBean Core instance from the toolbox.

```
$redbean = $toolbox->getRedBean();
```

If you do not want RedBean to alter the database structure anymore; you can either use:

```
$redbean->freeze();
```

or another kickstart method:

```
$toolbox = RedBean_Setup::kickstartFrozen( $dsn, $user, $pass );
```

Beans

RedBean simulates an object oriented database. The goal of RedBean is pretty simple; you give it an object and it stores the object for you. The objects you exchange with RedBean are not models but just plain beans. A bean can have many properties that are all public. A bean also has a type. To create a bean of a certain type you must ask RedBean to dispense such a bean for you:

```
$post = $redbean->dispense("post");
```

Now we have a post bean and we can populate this bean with all kinds of properties.

Rules for properties

There are some limitations concerning properties. First, the __info property is reserved for all meta information concerning the bean (like its type). Also the id property represents the primary key. A property may contain only primitive values.

Store a Bean in the database

To store a bean in the database we simply fill it:

```
$post->title = "My First Post";
$post->created = time();
```

And then we hand it over to RedBean to store it for us:

```
$id = $redbean->store( $post );
```

Now the beauty of RedBean is that it will create a post table for us automatically on the fly, as well as three columns (id, title and created) each with an appropriate column type and it will store the record for us in a readable way. It will also return the insert id.

Loading a Bean

Retrieving a bean from the database is even simpler. For instance, if we want to load the previously saved bean, we simply ask for it like this:

```
$post = $redbean->load("post",$id);
```

Now, we have got our (blog)post object back. There is not much to tell about this is there? We simply ask for a post with id $id and we get what we ask for.

Updating a Bean

Another beautiful feature of RedBean is that it seamlessly updates the database structure if needed; for instance if we decide to introduce a completely new property:

```
$post->rating = 5;
$redbean->store( $post );
```

RedBean just alters the table to make place for the new property by adding a column of the correct type.
We may even change the value from a column to a different type; of course in objects we do not really notice this, but RedBean will widen the column for us if we need more space:

```
$post->rating = "3 Stars";
$redbean->store( $post );
```

Deleting a Bean

To delete a bean from the database:

```
$redbean->trash( $post );
```

Importing and Exporting

RedBean has two service methods to facilitate ultra-fast bean loading. To import values from an array (i.e. POST):

```
$mybean->import($_POST,"intro,body,id");
```

This does the same as:

```
$mybean->intro = $_POST["intro"];
$mybean->body = $_POST["body"];
$mybean->id = $_POST["id"];
```

To export a bean to an array for use in a View object:

```
$aDullArray = $mybean->export();
```

Associations

Beans can be related; for instance a book has pages; a post may have comments etc. Associations can have qualities; they can form a tree structure or a circular structure. With RedBean you are free to design your own associative structures; to give you an idea how you can do this and to provide at least basic functionality RedBean ships with a very basic AssociationManager that can handle default associations.

Creating the Association Manager

The goal of the association manager is to manage associated beans. In order to do so, it needs a toolbox. To get an instance of the manager class use:

```
$a = new RedBean_AssociationManager( $toolbox );
```

Connecting two Beans

To create a connection between two beans, just associate them:

```
$a->associate($page, $user);
```

The $page and the $user are now associated. In RedBean you can associate anything with anything. All required tables will be generated on the fly for you.

Getting Related Beans

To get all pages associated with $user, type:

```
$keys = $a->related($user, "page" );
```

This will return an array containing the primary keys of every page related to $user. You can also get the page beans directly if you like. To accomplish this we need to use the batch loader from the RedBean Core class.

```
$pages = ($redbean->batch("page", $a->related($user, "page" )));
```

Breaking the connection between Beans

To clear the association between $page and $user type:
```
$a->unassociate($page, $user);
```
To clear all related pages:
```
$a->clearRelations($user, "page");
```

1 to N relations

From the RedBean perspective this is no more than a clear-all followed by a new relationship. To assign a page to a user use:
```
$a->set1toNAssoc($user, $page);
```
To re-assign to a different user we use:
```
$a->set1toNAssoc($user2, $page);
```
We can also assign multiple pages to the same user; but the second Bean can only have one of the first bean.

Trees

First Create a Tree Manager and give him the toolbox.
```
$tree = new RedBean_TreeManager( $toolbox );
```
To attach a child bean to a parent bean say:
```
$tree->attach( $parent, $child );
```
To get all the children under a parent:
```
$pages = $tree->children( $parent );
```
To get the parent id of a child bean, simply access the property parent_id:
```
$child->parent_id
```

Toolbox

The toolbox class in RedBean acts as a resource locator; its main function is to provide tools you often need. I could have stashed all functionality in one big object (oodb) but I want this library to be clean and maintainable. So I have given each class its own piece of logic and with that its own responsibility. A class in the RedBean realm has therefore only one reason to change. For you this means that the Kickstarter (who likes to make your job easier) returns a toolbox instead of a RedBean instance. Inside this toolbox you will find three classes that make up the core of RedBean.

The Adapter

The adapter is the class that communicates with the database for RedBean. This adapter makes it possible to execute queries to manipulate the database. To get an instance of this adapter use:
```
$adapter = $toolbox->getDatabaseAdapter();
```
For more information on the adapter see chapter: Queries.

The Query Writer

The Query Writer is only used by some RedBean modules to write platform specific SQL. You never have to use this in your own code but by providing the toolbox to other modules you give them the opportunity to take advantage of this system.

RedBean OODB

Most of the time you will need to interact with the RedBean Core Class instance. This object represents the object oriented database that RedBean as a whole tries to simulate. To pick this tool out of the toolbox say:

```
$redbean = $toolbox->getRedBean();
```

But if you have read the previous chapters carefully you would already have seen this :)

Finding Beans

This is where most ORM layers simply get it wrong. An ORM tool is only useful if you are doing object relational tasks. Searching a database has never been a strong feature of objects; but SQL is simply made for the job. In many ORM tools you will find statements like: $person->select("name")->where("age","20") or something like that. I found this a pain to work with. Some tools even promote their own version of SQL. To me this sounds incredibly stupid. Why should you use a system less powerful than the existing one? This is the reason that RedBean simply uses SQL as its search API.

Selecting Beans from the database

To search for specific beans in your database; we use a two-step system. First we select the IDs we want to fetch; then we let RedBean convert these to the appropriate beans. For instance, if we are looking for all pages that contain the name John we write:

```
$keys = $adapter->getCol("SELECT id FROM page WHERE `name` LIKE '%John%'");
```

Okay now we have a list of primary key IDs from the database. We can do anything with this list: chop it up for pagination; change the order etc. To convert them to beans we simply use the batch loader from the RedBean Core Class:

```
$pages = $redbean->batch("page", $keys);
```

RedBean Observers

RedBean supports observers to make it easy to add additional functionality without having to alter any class. To attach a listener to an object:

```
$redbean->addEventListener( $event, $myListener );
```

The following events are supported by the RedBean Core Class: "open" (load), "update" (store), "delete" (trash). The DBAdapter supports the event called "sql_exec". All observables will call the onEvent() method defined in the observer interface.

Queries

To perform a query, get a database adapter and :
```
$database->exec( "update page set title='test' where id=1" );
```
To fetch a multidimensional resultset directly after firing the query:
```
$database->get( "select * from page" );
```
To fetch a single row:
```
$database->getRow("select * from page limit 1");
```
To fetch a single column:
```
$database->getCol("select title from page");
```
To fetch a single cell:
```
$database->getCell("select title from page limit 1");
```
To get the latest insert-id:
```
$database->getInsertID();
```
To get the number of rows affected:
```
$database->getAffectedRows();
```
To escape a value or columnname for use in custom SQL use:
```
$database->escape( $value );
```
To get the original result resource to do your own processing:
```
$database->getRaw();
```

Transactions

From RedBean 0.7.8 on you can use transactions:
To start a transaction:
```
$database->startTransaction();
```
To commit:
```
$database->commit();
```
And finally, to roll back a transaction:
```
$database->rollback();
```

Meta Information

OODBBeans contain meta information; for instance the type of the bean. This information is hidden in a meta information field. You can use simple accessors to get and modify this meta information.

Fetching Meta Data

To get a meta property value:
```
$value = $bean->getMeta("my.property", $defaultIfNotExists);
```
The default default value is NULL.

Changing or Adding Meta Data

To set a meta property simply use a dot separated notation; preceding nodes do not have to exist; the system will create them automatically.

```
$bean->setMeta("type", "newtable"); //changes the table
```

To make Redbean add a unique index for several columns:

```
$bean->setMeta("buildcommand.unique.0", array( "column1", "column2",
"column3") );
```

RedShoe

RedShoe is a simple, tiny script for MySQL databases that updates existing databases to facilitate new RedBean software. For instance, if you deploy a RedBean program and you make some updates with RedBean this means you no longer have to compare the databases; RedShoe will do this for you and update the target database. It will add new columns (or widen existing ones), tables and indexes but it will not remove anything.

Download Redshoe

You can download RedShoe here.

Redshoe Usage

RedShoe is an easy to use commandline deployment tool for RedBean applications. But it can also be used for other non-redbean databases as well. To update or deploy a RedBean database on a remote server you need to register the database with RedShoe first.

Adding a database

To add a database for use with RedShoe issue the following command:

```
php redshoe.php add mysql localhost mydatabase root - db1
```

In this example, mydatabase is the name of the database to be added it will be assigned the internal name db1 (because you can have multiple databases with the same databasename on different hosts). The first argument is the database type to be used; only mysql is supported right now. Then, the second argument is the host or IP on wich your database resides, in our case this is localhost. Next comes the databasename followed by username and password. If you provide a dash (-) for password, this means: no password. The final argument is the internal redshoe name or label for this database.

Listing databases

To get an overview of the databases registered for use with redshoe invoke the following command:

```
php redshoe.php list
```

This command will list the databases currently registered; it will present you a list that looks somewhat like this:

```
db1: (mysql) 'mydatabase' on: localhost
```

Deployment

To perform a deployment with RedShoe type the following:

```
php redshoe.php deploy db1 production1 production2
```

The first database label indicates the source, the other database labels refer to the targets. This command will actually do nothing at all. It will just register that you want to deploy the contents of db1 to production1 and production2, that's all. You can list as many database targets as you want but only one source. The source database is always the first argument.

Exporting

To get the queries that are needed to update the production databases use:

```
php redshoe.php export
```

This will generate a file called import.sql for each production server in the destination list. Each file contains the required SQL to update that specific database to be compliant with the source database.

Distributing databases directly

Warning this feature should be used carefully and is not expected to work fully until version RedShoe 0.7 - This causes the deployment to be send to the target databases directly instead of saving it to SQL files. Syntax:

```
php redshoe.php distrib
```