Active Record Design Pattern

Rich Buggy

rich@buggy.id.au
http://www.buggy.id.au/

Copyright & Licence

About Design Patterns

- Design patterns are not new.
- You may be using them without knowing it.
- They describe a common problem and a repeatable solution.
- They are not language specific.
- They are not recipes.

Design Pattern Examples

- Breadcrumbs
- Singleton
- Factory
- MVC

What does Active Record solve?

- When creating models that interact with a database table you usually repeat a lot of code.
- Active Record applies the DRY principle to reduce the amount of code you need to write by providing a generic solution using CRUD.

CRUD!!

CRUD is an acronym for the actions most commonly performed on a rows of database table:

- **C**reate
- **R**etrieve
- **U**pdate
- **D**elete

CRUD in action

- A user registers with your site (CREATE).
- The user logins in (RETRIEVE).
- The user changes their password (UPDATE).
- Their account is removed (DELETE).

CRUD in code

```
class User {
    public function create() ...
    public static function findById($id) ...
    public static function findByUsername($username) ...
    public function update() ...
    public function delete() ...
}
```

The DRY principle

- What if you could remove the programming?
- It's possible by using convention instead of configuration

Objective

To make writing the model as easy as:

```
class User extends ActiveRecord {}
```

And using it as easy as:

```
$user = new User();

$user->username = 'fred';

$user->password = 'super-secret';

$user->create();
```

(or easier)

Table/Class rules

- The name of the class matches the name of the table in the database.
- Table names are always lower case with an underscore separating words.
- Class names are always upper camel case

Table/Class examples

Table	**Class**
user	User
cute_animal	CuteAnimal

Column/Property rules

- The name of the property matches the name of the column in the database.
- Column names are always lower case with an underscore separating words.
- Property names are always lower camel case.
- Any method `findByXxx()` is trying to retrieve records by matching column xxx with the value provided.

Column/Property examples

Column	**Property**
username	username
first_name	firstName
	findByUsername()
	findByFirstName()

Implicitly calling the parent constructor

- Two big changes occurred to classes in PHP 5:
  - Constructors are now always named `__construct()`
  - The constructor of the parent class is implicitly called if a class doesn't define it's own constructor
- This means the `ActiveRecord` class constructor will be called if we define our model as:

```
class User extends ActiveRecord {}
```

- Which allows us to put code to look up the column information into the `ActiveRecord` class constructor.

Undefined Properties

- PHP calls the following methods if you try to use a class property that hasn't been defined:

  `__get($property)`

  `__set($property, $value)`

  `__isset($property)` (PHP 5)

  `__unset($property)` (PHP 5)

- By defining these in the `ActiveRecord` class we can create properties at runtime that can be accessed like normal properties

Undefined Methods

- PHP calls the `__call($method, $params)` method if you try to call a method that hasn't been defined.
- Using this we can emulate the `findByXxx()` functions in `ActiveRecord`.

Create, Update and Delete

- Knowing the table and column names it's possible to generate SQL to INSERT, UPDATE and DELETE records in the table

What does Active Record look like?

```
class ActiveRecord
{
    public function __construct() ...

    public function __get($property) ...
    public function __set($property, $value) ...
    public function __isset($property) ...
    public function __unset($property) ...

    public function __call($method, $params) ...

    public function create() ...
    public function update() ...
    public function delete() ...
}
```

Creating a record

```
class User extends ActiveRecord {}

$user = new User();

$user->username = $_POST['username'];
$user->password = $_POST['password'];

$user->create();
```

Updating a record

```
class User extends ActiveRecord {}

$user = new User();

$user = $user->findById($_POST['id']);
$user->password = $_POST['password'];

$user->update();
```

Deleting a record

```
class User extends ActiveRecord {}

$user = new User();

$user = $user->findById($_POST['id']);

$user->delete();
```

But wait... there's more!!

- Let's add an optional parameter to `create()` and `update()`.
- It will be an associate array of property => value pairs
- When provided they will set to corresponding properties in the model before doing the INSERT or UPDATE
- We'll also add an optional parameter to `delete()` which is the value of the primary key field

Getting the array from the form

- Suppose you have a form that uses the POST method with the following fields

```
<input type="text" name="User[username]" />
<input type="password" name="User[password]" />
```

- These values can be accessed in PHP using

```
$_POST['Username']['username'];
$_POST['Username']['password'];
```

Creating becomes

```
class User extends ActiveRecord {}

$user = new User();
$user->create($_POST['User']);
```

Updating becomes

```
class User extends ActiveRecord {}

$user = new User();
$user = $user->findById($_POST['id']);
$user->update($_POST['User']);
```

Deleting becomes

```
class User extends ActiveRecord {}

$user = new User();
$user->delete($_POST['id']);
```

Issues

The main issue with Active Record is the requirement to query the database to determine the models structure. There are a number of ways of dealing with this.

- Ignore the problem (No caching)
- File caching
- memcache
- Freeze the model

Questions?