Relational Databases and Web Integration

Dr. Carl Pulley
c.j.pulley@hud.ac.uk

Simple PHP Example

- Simple blog application
  - allow the listing of blog entries
  - ability to show a given blog entry
- Blog entries will only have titles and content!
- Want to build application using MVC design pattern
  - will make extensive use of PHP objects and classes here!

Models

- Use PHP classes to interface to underlying database tables
- For each database table we'll have a PHP model class
  - table columns will be class properties

MySQL Table

```
create table entries (id int not null primary key, title varchar(255), content text);
```


Conventions:

- id will always be our primary key
- foreign keys will have a name ending in _id
- date columns will have a name ending in _at

Models

```
class Entry {
  public $id;
  public $title;
  public $content;

  function __construct($id) {
    $db = new PDO(..);
    $stmt = $db->prepare(“Select * from entries where id = ? limit 1”);
    $stmt->execute(array($id));
    $row = $stmt->fetch();
    $this->id = $row[‘id’];
    $this->title = $row[‘title’];
    $this->content = $row[‘content’];
    $db = null;
  }

  ..
}
```

Tuesday, 26 January 2010

Note: old style PHP used the class name as the name of the constructor function - avoid doing this since this feature will get depreciated at some point.
Constructors here simply de-serialise table rows to form a PHP object instance of the model.
PDO connection string should be something like the following on helios:
mysql:host=localhost;unix_socket=/usr/local/webstack1.5/var/run/mysql/mysql.sock;dbname=...

Models

```
class Entry {
  ..

  static function all() {
    $result = array();
    $db = new PDO(..);
    $stmt = “select id from entries”;
    foreach($db->query($stmt) as $row) {
      $result[] = new Entry($row[‘id’]);
    }
    $db = null;
    return $result;
  }

  static function find($id) {
    return new Entry($id);
  }
}
```


Model searching methods here are all static.

Controllers

```
class EntryController {
  private $view;

  function __construct($view) {
    $this->view = $view;
  }

  function index() {
    $this->view->index(Entry::all());
  }

  function show($id) {
    $this->view->show(Entry::find($id));
  }
}
```


Notice here how, after interacting with the underlying models, controllers setup a context of results (here modelled as method arguments) with which view code then renders an output.

Views

```
class EntryView {
  function index($entries) {
    foreach($entries as $entry) {
      echo “<h1>”.$entry->title.”</h1>\n”;
      echo “<p>”.$entry->content.”</p>\n”;
    }
  }

  function show($entry) {
    echo “<h1>”.$entry->title.”</h1>\n”;
    echo “<p>”.$entry->content.”</p>\n”;
  }
}
```


Notice how views get their code interpreted in a context (modelled here as parameters to the view method) setup by controllers.

Web Application

```
class Application {
  function __construct() {
    list($controller, $id) = split("/", $_SERVER['PATH_INFO'], 2);
    if ($id) {
      $action = 'show';
    } else {
      $action = 'index';
    }
    switch ($controller) {
      case 'entries':
        $controller = new EntryController(new EntryView());
        break;
    }
    switch ($action) {
      case 'index':
        $controller->index();
        break;
      case 'show':
        $controller->show($id);
        break;
    }
  }
}
```


Notice how applications route paths to locate the correct controller and action method within that controller.

Note: unfortunately, this type of solution fails to work in PHP! If it did work, then we could use restful paths like:

/entries

and:

/entries/42

Web Application

```
class Application {
  function __construct() {
    $controller = $_GET['controller'];
    $action = $_GET['action'];
    $id = $_GET['id'];
    switch ($controller) {
      case 'entries':
        $controller = new EntryController(new EntryView());
        break;
    }
    switch ($action) {
      case 'index':
        $controller->index();
        break;
      case 'show':
        $controller->show($id);
        break;
    }
  }
}
```

Tuesday, 26 January 2010

Notice here how we have to use unrestful paths like:
index.php?controller=entries&action=index
or:
index.php?controller=entries&action=show&id=42
here.

Isn't That a Lot of Code?

- For simple applications like this, using MVC appears to involve a lot of coding work!
  - sure, we have a flexible application
  - much of the code is boiler-plate code!
- By implementing scripts (with coding and naming conventions) we can automate much of this tedious work!!