2010

Đồ Án Cơ Sở: Tìm Hiểu PHP Framework CodeIgniter

ĐẠI HỌC KỸ THUẬT CÔNG NGHỆ THÀNH PHỐ HỒ CHÍ MINH
4/1/2010

Mục lục

1. Giới thiệu

1.1. Tổng quan

CodeIgniter là một nền tảng ứng dụng web nguồn mở được viết bằng ngôn ngữ PHP bởi Rick Ellis (CEO của EllisLab, Inc). Phiên bản đầu tiên được phát hành ngày 28.02.2006, phiên bản hiện tại: 1.7.2 (phát hành ngày 11.09.2009). Ý tưởng xây dựng CodeIgniter được dựa trên Ruby on Rails, một nền tảng ứng dụng web được viết bằng ngôn ngữ Ruby. Hiện tại, CodeIgniter đang được phát triển bởi ExpressionEngine Development Team thuộc EllisLab, Inc.

1.2. Những điểm nổi bật

Được thiết kế theo mô hình Model-View-Controller: Mô hình MVC giúp tách thành phần hiển thị giao diện (presentation) và xử lý (business logic) của một phần mềm thành những thành phần độc lập, từ đó giúp cho việc thiết kế, xử lý và bảo trì mã nguồn dễ dàng, đồng thời tăng khả năng mở rộng của phần mềm. CodeIgniter vận dụng mô hình này trong thiết kế, giúp tách biệt các tập tin giao diện với các tập tin xử lý dữ liệu, nâng cao khả năng quản lý và dễ bảo trì.

Nhỏ gọn: Gói cài đặt chỉ 404KB (không bao gồm phần User Guide). So với các PHP framework khác như CakePHP (1.3MB), Symfony (5.08MB) hay Zend Framework (5.66MB)…kích thước của CodeIgniter giúp giảm thiểu đáng kể không gian lưu trữ.

Tốc độ nhanh: CodeIgniter được đánh giá là PHP framework có tốc độ nhanh nhất hiện nay. Bằng cơ chế lưu nội dung vào bộ đệm (cache), kiểm tra bộ đệm trước khi tiến hành thực hiện yêu cầu, CodeIgniter giảm số lần truy cập và xử lý dữ liệu, từ đó tối ưu hóa tốc độ tải trang.

Miễn phí: CodeIgniter được phát hành dưới giấy phép Apache/BSD mở rộng, cho phép người dùng tự do thay đổi, phát triển và phân phối mã nguồn.

Hỗ trợ Search Engine Optimization: Cấu trúc URL của CodeIgniter rất thân thiện với các robot tìm kiếm.

Hệ thống thư viện phong phú: CodeIgniter cung cấp các thư viện phục vụ cho những tác vụ thường gặp nhất trong lập trình web, chẳng hạn như truy cập cơ sở dữ liệu, gửi email, kiểm tra dữ liệu, quản lý session, xử lý ảnh…đến những chức năng nâng cao như XML-RPC, mã hóa, bảo mật…

Bảo mật hệ thống: Cơ chế kiểm tra dữ liệu chặt chẽ, ngăn ngừa XSS và SQL Injection của CodeIgniter giúp giảm thiểu các nguy cơ bảo mật cho hệ thống.

1.3. Những điểm hạn chế

Chưa hỗ trợ Object-Relational Mapping: Object Relational Mapping (ORM) là một kỹ thuật lập trình, trong đó các bảng của cơ sở dữ liệu được ánh xạ thành các đối tượng trong chương trình. Kỹ thuật này giúp cho việc thực hiện các thao tác trong cơ sở dữ liệu (Create Read Update Delate – CRUD) dễ dàng, mã nguồn ngắn gọn hơn. Hiện tại, CodeIgniter vẫn chưa hỗ trợ ORM.

Chưa hỗ trợ AJAX: AJAX (Asynchronous JavaScript and XML) đã trở thành một phần không thể thiếu trong bất kỳ ứng dụng Web 2.0 nào. AJAX giúp nâng cao tính tương tác giữa người dùng và hệ thống, giúp cho người dùng có cảm giác như đang sử dụng ứng dụng desktop vì các thao tác đều diễn ra "tức thời". Hiện tại, CodeIgniter vẫn chưa có thư viện dựng sẵn nào để hỗ trợ xây dựng ứng dụng AJAX. Lập trình viên phải sử dụng các thư viện bên ngoài, như jQuery, Script.aculo.us, Prototype hay Mootools…

Chưa hỗ trợ một số module thông dụng: So sánh với framework khác, CodeIgniter không có các module thực thi một số tác vụ thường gặp trong quá trình xây dựng ứng dụng web như Chứng thực người dùng (User Authorization), Trình phân tích RSS (RSS Parser) hay Trình xử lý PDF…

Chưa hỗ trợ Event-Driven Programming: Event-Driven Programming (EDP) là một nguyên lý lập trình, trong đó các luồng xử lý của hệ thống sẽ dựa vào các sự kiện, chẳng hạn như click chuột, gõ bàn phím…Đây không phải là một khuyết điểm to lớn của CodeIgniter vì hiện tại, chỉ có một số ít framework hỗ trợ EDP, bao gồm Prado, QPHP và Yii.

1.4. Cài đặt

Yêu cầu hệ thống: CodeIgniter có thể hoạt động trên nhiều hệ điều hành và server, yêu cầu có cài đặt PHP phiên bản 4.x hoặc cao hơn; hệ quản trị cơ sở dữ liệu: MySQL (4.1+), MySQLi, Mircrosoft SQL Server, Postgres, Oracle, SQLite, và ODBC.

Hướng dẫn cài đặt:

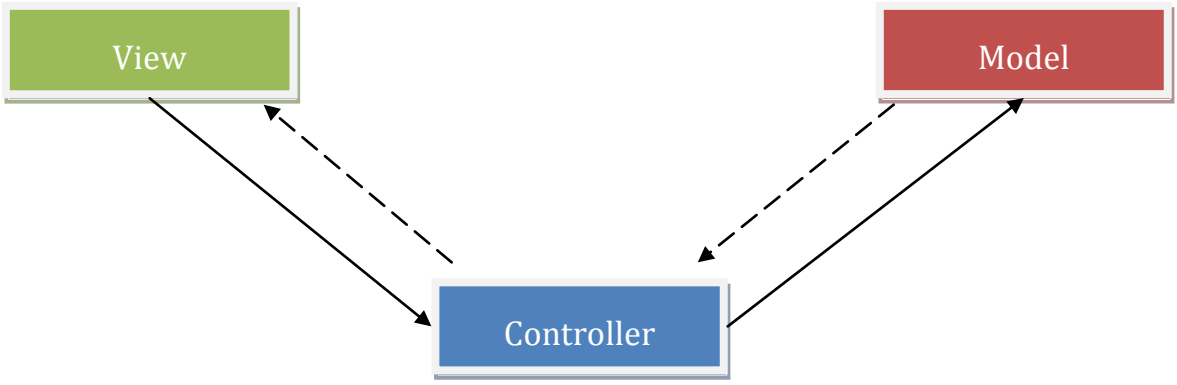
1. Download bộ nguồn CodeIgniter tại website http://codeigniter.com/, phiên bản hiện tại: 1.7.2.
2. Mở tập tin `application/config/config.php` bằng một chương trình soạn thảo, thay đổi giá trị `$config['base_url']`. Đây là đường dẫn tuyệt đối đến thư mục CodeIgniter trên server.
3. Nếu ứng dụng có tương tác với cơ sở dữ liệu, thiết lập các giá trị cần thiết trong tập tin `application/config/database.php`.
4. Upload tất cả thư mục và tập tin của CodeIgniter lên server.

Để nâng cao tính bảo mật hệ thống, người dùng có thể đổi tên thư mục system của CodeIgniter. Sau khi đổi tên, người dùng phải thay đổi giá trị biến `$system_folder` trong tập tin `index.php`.

2. Tìm hiểu

2.1. Mô hình Model-View-Controller

Model-View-Control (MVC) là một kiến trúc phần mềm, hiện đang được xem là một mẫu thiết kế trong công nghệ phần mềm. Mô hình MVC tách biệt phần xử lý dữ liệu ra khỏi phần giao diện, cho phép phát triển, kiểm tra và bảo trì các thành phần một cách độc lập.



Hình 1: *Minh họa mô hình MVC*

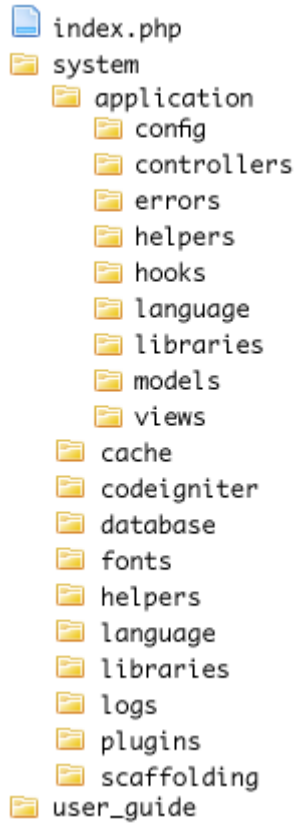
Theo đó:

- **Model** thể hiện các cấu trúc dữ liệu. Các lớp thuộc thành phần Model thường thực hiện các tác vụ như truy vấn, thêm, xóa, cập nhật dữ liệu. Khi dữ liệu trong Model thay đổi, thành phần View sẽ được cập nhật lại.
- **View** là thành phần thể hiện dữ liệu trong Model thành các giao diện tương tác với người sử dụng. Một Model có thể có nhiều View tùy thuộc vào các mục đích khác nhau.
- **Controller** đóng vai trò trung gian giữa Model và View. Thông tin người dùng từ View được gửi cho Controller xử lý, sau đó Controller tương tác với Model để lấy dữ liệu được yêu cầu, sau cùng Controller trả dữ liệu này về cho View.

Mô hình MVC thường được sử dụng trong các ứng dụng web, vì thành phần View (mã HTML/XHTML) được sinh ra từ các ngôn ngữ lập trình web. Thành phần Controller sẽ nhận các dữ liệu GET/POST, xử lý những dữ liệu này, sau đó chuyển sang Model xử lý.

Model sẽ trả dữ liệu về phía Controller, sau đó Controller sinh mã HTML/XHTML để thể hiện trên View.

2.2. Cấu trúc CodeIgniter



Hình 2: *Cấu trúc CodeIgniter*

Tập tin `index.php` được xem như controller đầu vào, tiếp nhận các yêu cầu từ phía client và chuyển các yêu cầu này cho hệ thống xử lý.

Thư mục `system` bao gồm phần lõi của CodeIgniter. Chúng bao gồm các thư viện xây dựng sẵn, các tập tin ngôn ngữ, ghi chú về hệ thống. Trong số đó, các thư mục sau khá quan trọng:

- Thư mục `application`: Dành cho lập trình viên, các tập tin được lập trình cho ứng dụng sẽ lưu trong thư mục này.
- Thư mục `cache`: Bộ đệm của hệ thống, chứa các trang đã được xử lý trước đó.
- Thư mục `helpers`: Chứa các hàm hỗ trợ cho lập trình viên khi viết ứng dụng.
- Thư mục `libraries`: Chứa các thư viện dựng sẵn của CodeIgniter.

Đối với lập trình viên, các tập tin của ứng dụng sẽ được lưu trong thư mục `system/application`. Trong đó:

- Thư mục `config`: Chứa các tập tin cấu hình hệ thống
- Thư mục `controllers`: chứa các lớp controller
- Thư mục `errors`: chứa các tập tin lỗi
- Thư mục `helpers`: chứa các hàm tiện ích do người dùng định nghĩa
- Thư mục `hooks`: chứa các tập tin để mở rộng mã nguồn CodeIgniter
- Thư mục `language`: chứa các tập tin ngôn ngữ
- Thư mục `libraries`: chứa các thư viện cho người dùng dùng định nghĩa
- Thư mục `models`: chứa các lớp model
- Thư mục `views`: chứa các lớp view

Ta cũng có thể đổi tên của thư mục `application` thành tên tùy ý. Sau khi đổi tên, cần thiết lập tên mới cho biến `$application_folder` trong tập tin `index.php`. Ngoài ra, ta cũng có thể tạo nhiều ứng dụng trong cùng một bộ cài đặt CodeIgniter bằng cách tạo ra các thư mục con bên trong `system/application`. Các thư mục này có cùng cấu trúc giống như thư mục `application` gốc.

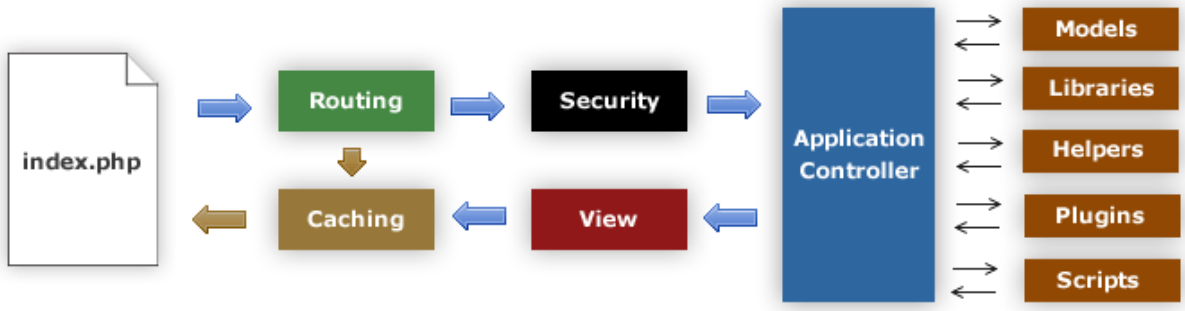
Chẳng hạn, ta có hai ứng dụng, `foo` và `bar`. Khi đó cấu trúc thư mục sẽ như sau:

```
system/application/foo/
system/application/foo/config/
system/application/foo/controllers/
system/application/foo/errors/
system/application/foo/libraries/
system/application/foo/models/
system/application/foo/views/
system/application/bar/
system/application/bar/config/
system/application/bar/controllers/
system/application/bar/errors/
system/application/bar/libraries/
system/application/bar/models/
system/application/bar/views/
```

Để biết tập tin index.php sẽ chạy ứng dụng nào, ta khai báo lại giá trị của biến `$application_folder`.

```
$application_folder = "application/foo";
```

2.3. Dòng chảy dữ liệu trong CodeIgniter



Hình 3: *Sơ đồ thể hiện dòng chảy dữ liệu trong CodeIgniter*

1. Tập tin `index.php` đóng vai trò làm controller đầu vào, thiết lập các tài nguyên cần thiết cho hệ thống.
2. *Routing:* Quá trình điều hướng giúp xác định các yêu cầu và hướng xử lý đối với chúng.
3. *Caching:* Nếu dữ liệu được yêu cầu đã được lưu trong bộ đệm, CodeIgniter sẽ trả dữ liệu trong bộ đệm về phía client. Quá trình xử lý kết thúc.
4. *Security:* Dữ liệu trước khi được chuyển đến các Controller sẽ được lọc để phòng chống XXS hoặc SQL Injection.
5. *Application Controller:* Controller xử lý dữ liệu nhận được bằng cách gọi đến các Models, Libraries, Helpers, Plugins…có liên quan.
6. *View:* Dữ liệu được chuyển qua View để hiển thị cho người dùng. Nếu chức năng caching được bật, dữ liệu sẽ được lưu trong cache cho những lần yêu cầu tiếp theo.

Trong thực tế, đôi khi ta cần xen vào các bước trong sơ đồ trên. Chẳng hạn, trước khi một controller được gọi, ta có thể muốn lưu log xuống tập tin trong hệ thống. CodeIgniter sử dụng cơ chế hook cho phép ta thực hiện việc này. Để kích hoạt hook, ta cần thay đổi giá trị của biến `$config['enable_hooks']` trong tập tin `application/config/config.php`:

```
$config['enable_hooks'] = TRUE;
```

Các hook được khai báo trong tập tin application/config/hooks.php, có mẫu sau:

```
$hook['pre_controller'] = array(
                                'class'    => 'MyClass',
                                'function' => 'Myfunction',
                                'filename' => 'Myclass.php',
                                'filepath' => 'hooks',
                                'params'   => array('beer', 'wine', 'snacks')
                                );
```

Trong đó:

- `class`: tên lớp sẽ được gọi. Nếu chỉ sử dụng một hàm thủ tục, ta có thể bỏ trống giá trị này.
- `function`: tên hàm/phương thức sẽ được gọi.
- `filename`: tên tập tin chứa lớp/hàm đó.
- `filepath`: đường dẫn đến tập tin hook. Nếu tập tin nằm trong thư mục `application/hooks`, ta chỉ cần đặt giá trị này bằng `hooks`. Nếu tập tin nằm trong thư mục `application/hooks/utilities`, ta đặt giá trị này bằng `hooks/utilities`. Lưu ý, không có dấu / ở cuối.
- `params`: các tham số được truyền vào cho hàm.

Khóa `pre_controller` cho biết thời điểm hook này được thực hiện. CodeIgniter hỗ trợ các thời điểm sau:

- `pre_system`: được gọi khi hệ thống vừa khởi động. Ở giai đoạn này chỉ có lớp benchmark và các hook được kích hoạt. Các tiến trình khác vẫn chưa xảy ra.
- `pre_controller`: được gọi trước khi các controller hoạt động. Các tiến trình routing và security đã được thực hiện.
- `post_controller_constructor`: được gọi ngay sau khi hàm tạo của controller được thi hành. Các hàm trong controller vẫn chưa được gọi.
- `post_controller`: được gọi ngay sau khi controller thực hiện xử lý yêu cầu.
- `display_override`: thực hiện việc chồng (override) hàm `__display()` để hoàn tất nội dung trang trước khi gửi đến trình duyệt của người dùng. Nội dung trang có thể được lấy bằng cách gọi `$this->CI->output->get_output()`.
- `cache_override`: thực hiện việc chồng hàm `_display_cache()` để lấy trang trong bộ đệm.
- `scaffolding_override`: không sử dụng từ CodeIgniter 1.6.
- `post_system`: được gọi sau khi hệ thống đã thực hiện xong một yêu cầu.

2.4. CodeIgniter URL

Theo mặc định, cấu trúc URL của CodeIgniter được thiết kế dựa vào các segment thay cho kiểu query truyền thống. Cách tiếp cận này giúp URL trở nên ngắn gọn, có ý nghĩa, dễ ghi nhớ và thân thiện với các bộ máy tìm kiếm. Một URL trong CodeIgniter có dạng:

```
domain.com/index.php/controller/method/param/…
```

Trong đó:

- Segment `controller` là tên của lớp controller được gọi.

- Segment `method` là tên của phương thức trong lớp controller ở trên.
- Segment `param` là các đối số của phương thức đó

Như vậy, URI `domain.com/index.php/product/view/1` sẽ mang ý nghĩa: Gọi đến phương thức `view()` thuộc lớp `Product` với đối số truyền vào bằng `1`.

Định tuyến URI (URI Routing)

Trong thực tế, URI `domain.com/index.php/product/view/1` lại không thân thiện với các bộ máy tìm kiếm bằng URI `domain.com/index.php/product/view/appple-ipad-1`. CodeIgniter cho phép lập trình viên có thể thay đổi cơ chế xử lý URL mặc định thông qua chức năng Định hướng URI bằng cách thiết lập các quy luật trong tập tin `application/config/routes.php`. Tập tin này chứa mảng `$route`, với khóa là URI mới và giá trị là URI cũ. Các quy luật được đọc theo thứ tự từ trên xuống, quy luật trước có độ ưu tiên cao hơn quy luật sau. CodeIgniter cho phép sử dụng các ký tự đại diện (wildcard) để thay thế. CodeIgniter đã quy định sẵn hai ký tự đại diện:

- `:num` tương ứng với các giá trị số.
- `:any` tương ứng với bất kỳ giá trị nào.

Ví dụ:

```
$route['product/view/(:num)'] = "product/view/$1";
```

Ý nghĩa: Gọi đến phương thức `view()` trong lớp `Product` với tham số truyền vào là giá trị kiểu số.

```
$route['product/:any'] = "product/find_product";
```

Ý nghĩa: Tất cả các yêu cầu bắt đầu bằng `product` sẽ gọi đến hàm `find_product()` của lớp `Product`.

Ta cũng có thể sử dụng biểu thức chính quy (Regular Expression) để thiết lập các quy luật. Chẳng hạn:

```
$route['product/([a-z]+)/(\d+)'] = "$1/id_$2";
```

Thêm hậu tố vào URL

Các bộ máy tìm kiếm có xu hướng thân thiện hơn với các tập tin HTML. Bằng cách thay đổi giá trị biến `$config['url_suffix']` trong tập tin `system/application/config/config.php`, ta có thể thêm hậu tố HTML hay bất cứ hậu tố nào khác vào cho URL. Ví dụ, với `$config['url_suffix'] = '.html'`, URL của ta sẽ như sau:

```
domain.com/index.php/controller/method/param.html
```

Loại bỏ chuỗi index.php trong URL

Theo mặc định, chuỗi `index.php` được thêm vào URL. Để loại bỏ chuỗi này, ta có thể sử dụng một tập tin `.htaccess` có nội dung như sau:

```
Options +FollowSymLinks All -Indexes
RewriteEngine On
RewriteCond $1 !^(index\.php|resources|robots\.txt)
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L,QSA]
```

2.5. Model

Model là những lớp được xây dựng nhằm thực hiện việc trao đổi thông tin với cơ sở dữ liệu. Một lớp Model có thể thực hiện các tác vụ truy vấn, thêm, xóa, cập nhật dữ liệu. Trong CodeIgniter, việc khai báo các lớp model dành cho một thực thể nào đó là không cần thiết, vì trong controller của thực thể đó, ta có thể gọi đến thư viện database để thực hiện các thao tác với cơ sở dữ liệu. Tuy nhiên, để tiện cho việc quản lý, xây dựng lớp model cho một thực thể được khuyến khích.

Một lớp model chuẩn trong CodeIgniter có cấu trúc như sau:

```
class Example_model extends Model {
    /**
     * Hàm tạo
     */
    function __contruct()
    {
        parrent::__contruct();
    }

    /**
     * Mô tả hàm
     * @param kiểu dữ liệu $param mô tả biến
     */
    function exampleFunction($param)
    {
        // Do something here
    }
    …
}
```

Khi khai báo một lớp model, ta cần tuân theo một số quy tắc:

- Tên lớp được viết hoa chữ đầu tiên, phần còn lại viết thường. Ví dụ: `User_model`, `Blog_model`, `Article_model`…

- Tên tập tin được đặt như tên lớp, và được đặt trong thư mục `application/models/`. Các thư mục có thể được lưu lồng vào nhau để thuận tiện cho việc quản lý. Ví dụ: `application/models/user_model.php`, `application/models/front_end/blog_model.php`, `application/models/back_end/article_model.php`…
- Bắt buộc phải kế thừa từ lớp `Model`. Trong hàm tạo của lớp con, phải gọi đến hàm tạo của lớp cha.

Để sử dụng model trong controller, ta sử dụng đoạn mã sau:

```
$this->load->model('model_name');
```

Trong đó, *`model_name`* là tên của lớp model. Nếu lớp model được lưu trong một thư mục, ta cần ghi rõ đường dẫn đến thư mục đó. Ví dụ lớp `Blog_model` được lưu trong thư mục `application/models/front_end/blog_model.php`. Để sử dụng lớp này, ta sử dụng đoạn mã sau:

```
$this->load->model('front_end/blog_model');
```

Sau khi load lớp model, ta có thể sử dụng lớp này trong chương trình bằng cách gọi

```
$this->model_name->method();
```

Để gán một tên khác cho lớp model khi sử dụng trong chương trình, ta có thể truyền vào tham số thứ hai như sau:

```
$this->load->model('model_name', 'ModelName');
```

Khi đó, để sử dụng các phương thức của lớp model, ta gọi:

```
$this->ModelName->method();
```

2.6. View

View là những tập tin HTML được xây dựng nhằm thể hiện dữ liệu trong model thành các giao diện tương tác với người dùng. View có thể là một trang web hoàn chỉnh, hay chỉ là một phần của trang web (header, footer, sidebar…). Nội dung của tập tin view, ngoài mã HTML còn có thể chứa mã PHP. View không bao giờ được gọi trực tiếp mà phải thông qua controller.

Để sử dụng view trong controller, ta sử dụng đoạn mã sau:

```
$this->load->view('view_name', $data);
```

Trong đó, *`view_name`* là tên của view, `$data` chứa các dữ liệu sẽ được hiển thị trong view. Cũng giống như model, ta có thể lưu view trong các thư mục để tiện cho việc quản lý.

Khi sử dụng, chỉ cần khai báo đường dẫn tương đối đến thư mục chứa view. Ví dụ tập tin `header.tpl.php` được lưu trong thư mục `application/views/front_end`. Để hiển thị view này, ta sử dụng đoạn mã sau:

```
$this->load->view('front_end/header.tpl');
```

CodeIgniter cho phép sử dụng nhiều view trong cùng một phương thức của controller. Dữ liệu trong các view sẽ được kết hợp lại với nhau. Ví dụ:

```
class Example extends Controller {
    function showExample()
    {
        // Loading multiple views
        $this->load->view('header.tpl');
        $this->load->view('content.tpl');
        $this->load->view('sidebar.tpl');
        $this->load->view('footer.tpl');
    }
}
```

Để hiển thị dữ liệu trong model ra view, ta có thể truyền một mảng hoặc đối tượng làm đối số thứ hai khi load view. Ví dụ:

```
$data = array(
    'name'  => 'Maxwell Smart',
    'email' => 'maxwell.smart@control.org',
    'age'   => '33'
);

$this->load->view('employee_detail', $data);
```

Hoặc

```
$objEmployee = new Employee();

$this->load->view('employee_detail', $objEmployee);
```

Khi đó, các khóa của mảng hoặc các thuộc tính của đối tượng sẽ được chuyển thành các biến để sử dụng trong tập tin view. Dưới đây là nội dung mẫu của tập tin `employee_detail.php`:

```
<p>Name: <?php echo $name ?></p>
<p>Email: <?php echo $email ?></p>
<p>Age: <?php echo $age ?></p>
```

Ta cũng có thể sử dụng các cấu trúc điều khiển (`if`, `else`, `switch`…case…), lặp (`for`, `while`, `do…while`…)…bên trong view để hiển thị dữ liệu. Ví dụ sau sẽ hiển thị danh sách các nhân viên:

```
// Employee Controller
$data['employeeList'] = $this->EmployeeModel->getAllEmployee();
```

```
$this->load->view('employee_list', $data);
```

Tập tin `employee_list.php`

```
<ol>
<?php
foreach( $employeeList as $objEmployee ) :
?>
<li>
    <p>Employee ID: <?php $objEmployee->employee_id ?></p>
    <p>Full-name: <?php $objEmployee->full_name ?></p>
    <p>D.O.B: <?php $objEmployee->dob ?></p>
    <p>Email: <?php $objEmployee->email ?></p>
</li>
<?php
endforeach;
?>
</ol>
```

CodeIgniter còn cho phép lập trình viên có thể lấy về nội dung của view thay vì xuất trực tiếp trình duyệt, bằng cách truyền `true` làm đối số thứ ba khi tải view.

```
$string = $this->load->view('myfile', '', true);
```

2.7. Controller

Controller là những lớp đóng vai trò trung gian giữa view và model. Controller nhận các yêu cầu từ phía người dùng, kiểm tra chúng trước khi chuyển qua cho model. Sau khi model xử lý yêu cầu và trả dữ liệu về, controller chuyển sang view để hiển thị dữ liệu cho người dùng. Trong CodeIgniter, các lớp controller được gán vào segment thứ nhất trong URI.

Một lớp controller chuẩn trong CodeIgniter có mẫu:

```
class Example extends Controller {
    /**
     * Hàm tạo
     */
    function __contruct()
    {
        parrent::__contruct();
    }

    /**
     * Mô tả hàm
     * @param kiểu dữ liệu $param mô tả biến
     */
    function method($param)
    {
        // Do something here
    }
    …
```

```
}
```

Khi khai báo một lớp controller, ta cần tuân theo một số quy tắc:

- Tên lớp được viết hoa chữ đầu tiên, phần còn lại viết thường. Ví dụ: `User`, `Blog`, `Article`…
- Tên tập tin được đặt như tên lớp, và được đặt trong thư mục `application/controllers/`. Các thư mục có thể được lưu lồng vào nhau để thuận tiện cho việc quản lý. Ví dụ: `application/controllers/user.php`, `application/controllers/front_end/blog.php`, `application/controllers/back_end/article.php`…
- Bắt buộc phải kế thừa từ lớp `Controller`. Trong hàm tạo của lớp con, phải gọi đến hàm tạo của lớp cha.

Segment thứ hai trong URI sẽ gọi đến phương thức tương ứng trong controller. Các giá trị của các segment còn lại trong URI chính là các tham số truyền vào cho phương thứ này. CodeIgniter quy định một phương thức đặc biệt, `index()`. Phương thức này có thể xem như đầu vào của controller, sẽ được tự động gọi trong trường hợp segment thứ hai của URI bị bỏ trống.

Ví dụ:

```
class Blog extends Controller {
    /**
     * Hàm tạo
     */
    function __contruct()
    {
        parrent::__contruct();
    }

    /**
     * Hàm đầu vào
     */
    function index()
    {
        echo 'Welcome to my blog';
    }

    /**
     * Hiển thị nội dung bài viết
     * @param int $entryId ID của bài viết
     */
    function view($entryId)
    {
        // Code hiển thị nội dung bài viết
    }
}
```

Khi truy xuất đến URI `domain.com/index.php/blog`, phương thức index() được gọi, ta sẽ thấy chuỗi Welcome to my blog. Tiếp theo, truy xuất đến URI `domain.com/index.php/blog/view/1` sẽ hiển thị nội dung của bài viết có ID bằng 1.

Ngoài ra, CodeIgniter cho phép lập trình thay đổi cơ chế gọi phương thức thông qua segment thứ hai bằng hàm `_remap()`. Hàm này sẽ được quyết định cụ thể những phương thức nào sẽ được gọi tương ứng với từng segment.

```
function _remap($method)
{
    if ($method == 'foo')
    {
        $this->bar();
    }
    else
    {
        $this->default_method();
    }
}
```

Lưu ý: các phương thức private trong controller được bắt đầu bằng ký tự gạch dưới (_), ví dụ: `_remap()`, `_my_private_method()`...Các hàm này không thể được gọi bằng segment thứ hai trong URI.

2.8. Các thư viện

Sức mạnh của CodeIgniter nằm ở các thư viện xây dựng sẵn. Hiện tại, CodeIgniter hỗ trợ người dùng 26 thư viện sau:

Tên thư viện	Chức năng
Benchmarking	Hỗ trợ đánh giá hiệu năng hệ thống
Calendar	Hỗ trợ tạo lịch tự động
Cart	Hỗ trợ chức năng giỏ hàng trong các website thương mại điện tử
Config	Cho phép thiết lập hệ thống
Database	Hỗ trợ thao tác trên cơ sở dữ liệu
Email	Hỗ trợ gửi email
Encryption	Hỗ trợ mã hóa và giải mã thông tin

File Uploading	Hỗ trợ upload tập tin từ người dùng lên server
Form Validation	Cho phép kiểm tra dữ liệu người dùng
FTP	Hỗ trợ kết nối FTP
HTML Table	Hỗ trợ xây dựng bảng tự động
Image Manipulation	Hỗ trợ xử lý ảnh
Input and Security	Hỗ trợ xử lý dữ liệu đầu vào và bảo mật
Loader	Hỗ trợ tải các thành phần của CodeIgniter
Language	Hỗ trợ hiển thị đa ngôn ngữ
Output	Hỗ trợ xuất thông tin về phía trình duyệt của người dùng
Pagination	Hỗ trợ phân trang tự động
Session	Hỗ trợ xử lý session
Trackback	Cho phép và nhận thông tin trackback
Template Parser	Cho phép xây dựng và xử lý mã giả trong các tập tin view
Typography	Hỗ trợ định dạng văn bản
Unit Testing	Hỗ trợ unit testing
URI	Cho phép lấy thông tin từ URI
User Agent	Cho phép xác định thông tin trình duyệt của người dùng, thiết bị di động hay các robot đang truy cập website
XML-RPC	Cho phép gửi yêu cầu đến một XML-RPC hoặc tự xây dựng một XML-RPC cho hệ thống
Zip Encoding	Cho phép tạo tập tin ZIP

Để sử dụng một thư viện nào đó, ta khai báo như sau:

```
$this->load->library('lib_name');
```

Trong đó, *lib_name* là tên của thư viện. Ví dụ, để sử dụng thư viện Form Validation, ta khai báo:

```
$this->load->library('form_validation');
```

Sau khi khai báo, ta có thể truy xuất đến các phương thức của thư viện bằng cách gọi:

```
$this->lib_name->method();
```

Thêm một thư viện mới

CodeIgniter cho phép lập trình viên tạo các thư viện mới, mở rộng hoặc thay thế các thư viện sẵn có. Các thư viện do lập trình viên tạo ra sẽ được lưu trong thư mục `application/libraries`. **Lưu ý**: thư viện Database và Loader không thể mở rộng hoặc thay thế.

Khi khai báo một lớp thư viện, cần tuân theo các quy tắc sau:

- Tên tập tin phải được viết hoa, ví dụ: `Someclass.php`
- Khi khai báo lớp, tên lớp phải được viết hoa ký tự đầu tiên, ví dụ: `class Someclass`. Tên lớp và tên tập tin phải trùng nhau.

Để có thể sử dụng các thư viện, helper, plugin của CodeIgniter trong thư viện mới, ta không thể dùng khai báo `$this->load->xxx()` vì lớp thư viện mới là một lớp độc lập và không có thuộc tính `load`. Để giải quyết vấn đề này, ta sẽ khai báo như sau:

```
$CI =& get_instance();
```

Sau khi đã có đối tượng $CI, ta có thể khai báo sử dụng tài nguyên của CodeIgniter như bình thường.

```
$CI->load->helper('url');
$CI->load->library('session');
$CI->config->item('base_url');
```

Lưu ý: không sử dụng phương thức `get_instance()` trong hàm tạo nếu server đang chạy PHP4.

Đây là khai báo mẫu một thư viện `Someclass`

```
<?php
class Someclass {
    private $ci;

    function __construct()
    {
        $this->ci =& get_instance();
```

```
        $this->ci->load->library('form_validation');
        $this->ci->load->helper( array('url', 'date') );
        $this->ci->load->database();
    }

    function doSomething()
    {
        // Code goes here
    }
}
```

Thay thế một thư viện

Để thay thế một thư viện có sẵn của CodeIgniter, ta khai báo một lớp mới trùng tên với lớp thư viện, sau đó lưu lớp này vào thư mục `application/libraries`. Chẳng hạn, để thay thế lớp Email, ta tạo tập tin `application/libraries/Email.php` và khai báo:

```
class CI_Email {

}
```

Lưu ý: Các lớp thư viện của CodeIgniter đều được bắt đầu bằng tiền tố `CI_`.

Mở rộng một thư viện

Để mở rộng một thư viện, ta khai báo lớp mới kế thừa từ lớp thư viện cần mở rộng, và đặt tên lớp này với tiền tố `MY_` (hoặc giá trị của biến `$config['subclass_prefix']` trong tập tin `application/config/config.php`). Ví dụ, để mở rộng lớp Email, ta tạo một tập tin `application/libraries/MY_Email.php` và khai báo:

```
class MY_Email extends CI_Email {

}
```

Khi cần sử dụng thư viện Email, ta dùng:

```
 $this->load->library('email'); // Không có tiền tố MY_
```

2.9. Các helper và plugin

Helper là tập hợp những hàm tiện ích được xây dựng nhằm hỗ trợ lập trình viên thực hiện một số công việc nào đó. Chẳng hạn, URL Helper giúp tạo liên kết, Form Helper giúp tạo form, Cookie Helper giúp xử lý cookie…Các helper không được xây dựng thành từng lớp đối tượng, đơn giản chúng là tập hợp những hàm thủ tục được phân thành từng nhóm riêng biệt, và chúng không phụ thuộc vào nhau.

Các helper của CodeIgniter được lưu trong thư mục `system/helpers`. Lập trình viên có thể tự xây dựng riêng các helper cho mình, hoặc sử dụng helper được chia sẻ trên

mạng. các helper này được lưu trong thư mục `system/application/helpers`. Khi khai báo sử dụng một helper nào đó, trước tiên CodeIgniter sẽ tìm trong thư mục `system/application/helpers`, nếu không tìm thấy sẽ chuyển sang tìm trong thư mục `system/helpers`.

Để sử dụng một helper, ta khai báo như sau:

```
$this->load->helper('helper_name');
```

Trong đó, *`helper_name`* là tên của helper, bỏ đi phần ".php" hoặc "_helper". Chẳng hạn để sử dụng URL Helper, là tập tin có tên `url_helper.php`, ta khai báo:

```
$this->load->helper('url');
```

Để sử dụng nhiều helper cùng lúc, ta có thể khai báo như sau

```
$this->load->helper( array('helper1', 'helper2', 'helper3') );
```

Một helper có thể được khai báo ở bất cứ đâu, thậm chí trong một tập tin view. Nhưng tốt hơn là nên khai báo helper trong các lớp controller. Sau khi khai báo, ta có thể sử dụng các hàm của helper. CodeIgniter cho phép lập trình viên mở rộng các helper sẵn có bằng cách tạo một tập tin trùng tên với tên của helper cần mở rộng, và có thêm tiền tố `MY_`. Tiền tố này có thể được thay đổi bằng cách thiết lập giá trị biến `$config['subclass_prefix']` trong tập tin `application/config/config.php`. Lưu ý: `CI_` là tiền tố mặc định của CodeIgniter, ta không nên sử dụng.

Ví dụ, để thêm một số hàm chức năng cho Array Helper, ta tạo tập tin `MY_array_helper.php` trong thư mục `system/application/helpers`, sau đó khai báo như sau:

```
<?php
// any_in_array() không có trong Array Helper, khai báo này sẽ
// tạo hàm mới
function any_in_array($needle, $haystack)
{
    $needle = (is_array($needle)) ? $needle : array($needle);

    foreach ($needle as $item)
    {
        if (in_array($item, $haystack))
        {
            return TRUE;
        }
    }

    return FALSE;
}

// random_element() là hàm có sẵn trong Array Helper, khai báo này sẽ
```

```
// chồng lên hàm cũ
function random_element($array)
{
    shuffle($array);
    return array_pop($array);
}
```

Plugin có chức năng tương tự như helper. Điểm khác biệt là plugin thường chỉ có duy nhất một hàm, trong khi helper là tập hợp các hàm cùng thực hiện một loại tác vụ nào đó. Các plugin được lưu trong thư mục `system/plugins`, hoặc `system/application/plugins`. Khi sử dụng plugin, CodeIgniter sẽ tìm trong thư mục `system/application/plugins` trước, sau đó đến thư mục `system/plugins`.

Để sử dụng plugin, ta khai báo như sau:

```
$this->load->plugin('plugin_name');
```

Trong đó, *`plugin_name`* là tên của helper, bỏ đi phần ".php" hoặc "plugin". Chẳng hạn để sử dụng Captcha plugin, là tập tin có tên `captcha_pi.php`, ta khai báo:

```
$this->load->plugin('captcha');
```

Ta cũng có thể sử dụng nhiều plugin cùng lúc bằng cách khai báo:

```
$this->load->plugin( array('plugin1', 'plugin2', 'plugin3') );
```

2.10. Tự động khai báo

CodeIgniter hỗ trợ chức năng cho phép tự động khai báo sử dụng các helper, plugin, library, model, các tập tin ngôn ngữ và các tập tin cấu hình mỗi khi hệ thống hoạt động. Để sử dụng chức năng này, ta thiết lập trong tập tin `system/application/config/autoload.php`. Ví dụ, để tự động sử dụng URL Helper, Form Helper, ta thiết lập như sau:

```
$autoload['helper'] = array('url', 'form');
```

2.11. Quản lý lỗi

CodeIgniter cung cấp một số hàm để quản lý lỗi xảy ra trong hệ thống. Theo mặc định, CodeIgniter sẽ hiển thị tất cả các lỗi xảy ra trong quá trình hệ thống hoạt động. Để thiết lập từng loại lỗi nào được hiển thị, ta có thể thay đổi tham số của hàm `error_reporting()` trong tập tin `index.php`. CodeIgniter còn có cơ chế cho phép ghi lỗi thành tập tin văn bản để thuận tiện cho việc lưu trữ cũng như sửa lỗi.

CodeIgniter hỗ trợ các hàm sau để xử lý lỗi:

```
show_error($message [, int $status_code = 500 ] )
```

Hàm `show_error()` sẽ hiển thị thông báo lỗi với mẫu là tập tin `application/errors/error_general.php`. Biến tùy chọn `$status_code` cho phép thiết lập mã trạng thái HTTP của lỗi này, mặc định là 500 (Internal Server Error).

```
show_404($path)
```

Hàm `show_404()` sẽ hiển thị thông báo không tìm thấy tài nguyên yêu cầu với mẫu là tập tin `application/errors/error_404.php`. `$path` cho biết đường dẫn của tài nguyên. Hàm này sẽ được tự động gọi khi hệ thống không tìm thấy controller.

```
log_message($level, $message)
```

Hàm `log_message()` cho phép lưu các thông báo xuống tập tin văn bản. Có 3 loại thông báo:

1. Thông báo lỗi (`$level = 'error'`): Đây là những lỗi thực sự xảy ra trong hệ thống, có thể là lỗi của PHP hoặc của người dùng.
2. Thông báo gỡ lỗi (`$level = 'debug'`): Đây là những thông báo hỗ trợ cho việc tìm và sửa lỗi. Chẳng hạn, khi một lớp được khởi tạo, ta có thể lưu thông tin này lại để giúp cho việc gỡ lỗi sau này.
3. Thông báo chung (`$level = 'info'`): Cấp độ thấp nhất, cung cấp thông tin đơn giản, đôi khi mang tính chất ghi chú.

Lưu ý: Để có thể ghi xuống tập tin văn bản, thư mục `system/logs` phải được cho phép ghi (writable).

Ví dụ:

```
if ($some_var == "")
{
    log_message('error', 'Some variable did not contain a value.');
}
else
{
    log_message('debug', 'Some variable was correctly set');
}

log_message('info', 'The purpose of some variable is to provide some value.');
```

2.12. Lưu trữ bộ đệm

Lưu trữ bộ đệm (caching) giúp tối ưu hóa hiệu suất của hệ thống. Dù cho tốc độ xử lý của CodeIgniter khá nhanh, nhưng khi thực hiện các thao tác truy xuất dữ liệu, CodeIgniter vẫn phải sử dụng các tài nguyên của hệ thống như bộ nhớ, dung lượng

đĩa…Bằng cách kích hoạt chức năng lưu trữ vào bộ đệm, tốc độ tải trang của CodeIgniter có thể tương đương với các trang web tĩnh.

Về căn bản, tất cả các trang đều được lưu trong bộ đệm. Lập trình viên có thể thiết lập thời gian tồn tại của trang trong bộ đệm trước khi trang được tái tạo nội dung. Khi trang được tải lần đầu tiên, nội dung của trang được lưu vào thư mục `system/cache`. Trong những lần tải sau, nội dung của trang được lấy ra và gửi đến trình duyệt của người dùng. Sau một khoảng thời gian định trước, nội dung này được xóa và khởi tạo lại.

Để bật chức năng lưu trữ bộ đệm cho một trang, ta khai báo như sau trong controller của trang đó:

```
$this->output->cache($n);
```

Với `$n` là số phút trang sẽ được lưu trong bộ đệm. Để hủy chức năng lưu trữ bộ đệm, ta chỉ cần xóa đoạn mã ở trên. Các nội dung đã được lưu trong bộ đệm trước đó sẽ tự động xóa sau khoảng thời gian `$n`.

Lưu ý: Để có thể ghi vào bộ đệm, thư mục `system/cache` phải được cho phép ghi (writable).

2.13. Debugging

Tìm và sửa lỗi là công việc thường gặp trong quá trình xây dựng hệ thống. CodeIgniter cung cấp cho lập trình viên lớp Profiler, giúp theo dõi dữ liệu `$_POST` gửi lên, các câu truy vấn cũng như các kết quả đo lường về CPU, bộ nhớ. Lớp Profiler được tự động kích hoạt khi hệ thống hoạt động, nên ta không cần phải khai báo khởi tạo lớp này.

Để bật profiler, ta sử dụng đoạn mã sau ở bất cứ đâu trong controller:

```
$this->output->enable_profiler(TRUE);
```

Khi đó, thông tin profiler sẽ được hiển thị ở cuối trang. Profiler cung cấp cho lập trình viên thông tin về URI, controller và phương thức được gọi, dung lượng bộ nhớ sử dụng, thông tin kiểm chuẩn, dữ liệu GET, POST và các câu truy vấn.

Để tắt profiler, ta sử dụng đoạn mã sau:

```
$this->output->enable_profiler(FALSE);
```

2.14. Bảo mật

Cơ chế bảo mật chặt chẽ của CodeIgniter giúp lập trình viên có thể yên tâm khi xây dựng ứng dụng. Để phòng ngừa các phương thức tấn công phổ biến như XSS hay SQL Injection, CodeIgniter chỉ cho phép các ký tự sau xuất hiện trong URI:

- Dữ liệu kiểu số và chữ
- Dấu ngã ~, dấu chấm (.), dấu hai chấm (:), dấu gạch ngang (-), dấu gạch dưới (_)

Bằng cách này, các mã độc không thể được truyền trực tiếp vào hệ thống. Mặc định, CodeIgniter không chấp nhận dữ liệu GET vì cấu trúc URL của CodeIgniter dựa trên segment thay cho dạng query truyền thống.

Trong quá trình khởi động hệ thống, tất cả các biến toàn cục của hệ thống đều bị hủy, ngoại trừ `$_POST` và `$_COOKIE`. Cũng trong quá trình này, giá trị `magic_quotes_runtime` trong tập tin `php.ini` cũng được gán giá trị `off`. Điều này giúp cho lập trình viên không phải lọc các ký tự escape khi đọc thông tin từ cơ sở dữ liệu.

Nguyên tắc lập trình:

Trước khi đưa bất kỳ dữ liệu nào vào xử lý trong hệ thống, dù cho đó là thông tin được gửi từ phía người dùng, cookie, giá trị URI, dữ liệu XML-RPC hay thậm chí là giá trị của biến `$_SERVER`, lập trình viên được khuyến khích thực hiện các nguyên tắc sau:

1. Lập trình viên cần quan niệm "All Input Data Is Evil", dữ liệu cần phải được lọc trước khi xử lý. CodeIgniter cung cấp thư viện Input and Security để lọc dữ liệu.
2. Kiểm tra tính toàn vẹn của dữ liệu (kiểu dữ liệu, kích thước, độ dài…). Đôi khi quá trình này được thay thế bởi bước 1. CodeIgniter cung cấp thư viện Form Validation giúp cho việc kiểm tra dữ liệu người dùng.
3. Escape dữ liệu trước khi đưa vào lưu trữ trong cơ sở dữ liệu.

3. Những thư viện chính

3.1. Input and Security

Thư viện Input and Security được xây dựng với mục đích:

- Tiền xử lý dữ liệu hệ thống (các biến `$_POST`, `$_SERVER`, `$_COOKIE`…) nhằm loại bỏ các mã độc đính kèm.
- Cung cấp một số hàm helper để thu thập dữ liệu nhập vào và xử lý chúng.

Lớp Input and Security được khởi tạo một cách tự động khi hệ thống hoạt động, do đó lập trình viên không cần phải khai báo khởi tạo lớp này. Mỗi khi một controller được gọi, lớp Input thực hiện các hành động sau:

- Hủy biến `$_GET`.
- Hủy tất cả các biến toàn cục của hệ thống trong trường hợp `register_globals = on`.
- Lọc khóa của các biến `$_POST`, `$_COOKIE`, chỉ cho phép khóa là các ký tự số, chữ và một số ký tự khác.
- Lọc XSS.
- Chuẩn hóa ký tự xuống dòng thành `\n`.

3.1.1. Cơ chế lọc XSS

Cross Site Scripting (XSS) là một kỹ thuật tấn công thông dụng bằng cách chèn JavaScript hoặc các ngôn ngữ khác vào dữ liệu gửi lên hệ thống, nhằm mục đích đánh cắp cookie hay thực hiện các hành vi nguy hại khác. CodeIgniter cung cấp cơ chế lọc dữ liệu nhằm loại bỏ các đoạn mã này. Nếu phát hiện mã độc, CodeIgniter sẽ chuyển chúng thành các thực thể ký tự (character entities). Lập trình viên có thể kích hoạt cơ chế này hoạt động trên toàn hệ thống hoặc chỉ áp dụng trong từng trường hợp cụ thể. Theo mặc định, chức năng này được tắt trên toàn hệ thống vì nó tiêu tốn nhiều tài nguyên khi thực hiện. Nếu muốn kích hoạt, ta thay đổi giá trị của biến `$config['global_xss_filtering']` trong tập tin `application/config/config.php`.

```
$config['global_xss_filtering'] = TRUE;
```

Lập trình viên được khuyến khích sử dụng chức năng này trong từng trường hợp cụ thể bằng cách gọi hàm:

```
$this->input->xss_clean(mixed $var [, boolean $isImage])
```

Trong đó, `$var` là biến cần được lọc XSS. Biến tùy chọn `$isImage` được sử dụng trong trường hợp người dùng upload một tập tin lên hệ thống. Khi đó, hàm `xss_clean()` sẽ kiểm tra tập tin được upload có bị nhúng mã độc không. Hàm trả về `TRUE` nếu tập tin an toàn, ngược lại trả về `FALSE`.

3.1.2. Các hàm tiện ích

Lớp Input and Security cung cấp một số hàm tiện ích, giúp lập trình viên có thể lấy dữ liệu từ phía người dùng an toàn.

```
$this->input->post(string $key [, boolean $xssFilter])
```

Hàm `post()` sẽ lấy dữ liệu POST từ phía người dùng, tương tự như sử dụng biến `$_POST[$key]`. Tuy nhiên, hàm này sẽ trả về `FALSE` nếu dữ liệu cần lấy không tồn tại. Biến tùy chọn `$xssFilter` khi được truyền vào `TRUE` sẽ kích hoạt chức năng lọc XSS cho dữ liệu này.

```
$this->input->get(string $key [, boolean $xssFilter])
```

Tương tự, hàm `get()` sẽ lấy dữ liệu GET từ phía người dùng. Hàm trả về `FALSE` nếu dữ liệu không tồn tại.

```
$this->input->get_post(string $key [, boolean $xssFilter])
```

Hàm `get_post()` sẽ tìm dữ liệu trong mảng `$_POST` trước, nếu không tìm thấy sẽ chuyển sang tìm trong mảng `$_GET`. Hàm trả về `FALSE` nếu không tìm thấy dữ liệu.

```
$this->input->cookie(string $key [, boolean $xssFilter])
```

Hàm `cookie()` sẽ lấy dữ liệu từ mảng `$_COOKIE`. Hàm trả về `FALSE` nếu dữ liệu không tồn tại.

```
$this->input->server(string $key [, boolean $xssFilter])
```

Hàm `server()` sẽ lấy dữ liệu từ mảng `$_SERVER`. Hàm trả về `FALSE` nếu dữ liệu không tồn tại.

```
$this->input->ip_address()
```

Hàm `ip_address()` sẽ trả về địa chỉ IP của người sử dụng. Nếu địa chỉ IP không hợp lệ, hàm sẽ trả về chuỗi `0.0.0.0`

```
$this->input->valid_ip(string $ip)
```

Kiểm tra địa chỉ IP truyền vào có hợp lệ không, trả về `TRUE` nếu địa chỉ hợp lệ, ngược lại hàm trả về `FALSE`.

```
$this->input->user_agent()
```

Trả về thông tin trình duyệt của người sử dụng. Nếu thông tin này không có, hàm trả về `FALSE`.

3.2. Form Validation

Thư viện Form Validation của CodeIgniter giúp lập trình viên kiểm tra dữ liệu được gửi lên từ phía người dùng. Trong thư viện Form Validation đã xây dựng sẵn một số ràng buộc dữ liệu thường gặp, ta có thể áp dụng vào lập trình một cách dễ dàng. Để sử dụng thư viện này, ta khai báo như sau:

```
$this->load->library('form_validation');
```

Sau khi khai báo, ta có thể sử dụng các phương thức của thư viện này bằng cách sử dụng đối tượng `$this->form_validation`.

3.2.1. Thiết lập các điều kiện kiểm tra

Bằng cách kết hợp các điều kiện có sẵn và các hàm tự định nghĩa, thư viện Form Validation giúp lập trình viên có thể kiểm tra dữ liệu nhập vào từ phía người dùng. Để sử dụng các điều kiện này, ta khai báo như sau:

```
$this->form_validation->set_rules(string $field, string $label, string $rules);
```

Trong đó:

- `$field` là tên của trường HTML được áp dụng điều kiện này, thông thường là giá trị thuộc tính `name` của tag `INPUT`, `TEXTAREA`, `SELECT`...Lưu ý: Nếu sử dụng mảng làm tên trường (thường gặp đối với checkbox, list...), ví dụ `chkColor[]`, ta phải truyền chính xác tên trường, kể cả ký tự `[]`.
- `$label` là tên của trường đó. Giá trị của biến này sẽ được sử dụng trong các thông báo lỗi.
- `$rules` là chuỗi chứa các điều kiện kiểm tra. Các điều kiện này có thể được xây dựng sẵn bởi CodeIgniter, hoặc là các hàm nhận một đối số của PHP hay các hàm callback do lập trình viên tự định nghĩa. Các điều kiện cách nhau bởi ký tự gạch đứng |.

Xét ví dụ sau:

```
$this->form_validation->set_rules('txtUsername', 'Username',
'trim|required|min_length[5]|max_length[12]|xss_clean');
```

Điều kiện trên có nghĩa là, trường `txtUsername` là trường bắt buộc (`required`), giá trị của trường có chiều dài tối thiểu là 5 ký tự (`min_length[5]`), chiều dài tối đa là 12 ký tự (`max_length[12]`), dữ liệu này sẽ được loại bỏ khoảng trắng ở hai đầu (`trim`) và lọc XSS (`xss_clean`).

Các điều kiện có sẵn trong thư viện Form Validation:

Tên luật	Mô tả	Ví dụ
`required`	Trả về `FALSE` nếu trường rỗng	
`matches`	Trả về `FALSE` nếu giá trị của trường không trùng với giá trị của trường được	`matches[txtPassword]`

	truyền vào	
min_length	Trả về FALSE nếu chiều dài giá trị của trường ít hơn số ký tự quy định	min_length[5]
max_length	Trả về FALSE nếu chiều dài giá trị của trường nhiều hơn số ký tự quy định	max_length[12]
exact_length	Trả về FALSE nếu chiều dài giá trị của trường không đúng bằng số ký tự quy định	exact_length[8]
alpha	Chỉ cho phép giá trị của trường chứa các ký tự chữ	
alpha_numeric	Chỉ cho phép giá trị của trường chứa các ký tự chữ và số	
alpha_dash	Chỉ cho phép giá trị của trường chứa các ký tự chữ, số, dấu gạch ngang (-) và dấu gạch dưới (_)	
numeric	Chỉ cho phép giá trị của trường chứa các ký tự số	
integer	Chỉ cho phép trường chứa giá trị số nguyên	
is_natural	Chỉ cho phép trường chứa giá trị số tự nhiên	
is_natural_no_zero	Chỉ cho phép trường chứa giá trị số tự nhiên, bỏ số 0	
valid_email	Trả về FALSE nếu giá trị của trường không phải là một địa chỉ email hợp lệ	
valid_emails	Trả về FALSE nếu giá trị của trường không phải là một tập hợp các địa chỉ email hợp lệ, ngăn cách bởi dấu phẩy (,)	
valid_ip	Trả về FALSE nếu giá trị của trường không phải là một địa chỉ IP hợp lệ	
valid_base64	Trả về FALSE nếu giá trị của trường chứa các ký tự không phải là các ký tự của mã hóa Base 64	

Các hàm xử lý dữ liệu trong thư viện Form Validation:

Tên hàm	Mô tả
xss_clean	Lọc XSS từ dữ liệu gửi lên
prep_for_form	Chuyển đổi mã HTML thành các thực thể ký tự (character entites) để hiển thị chính xác trên các tag như INPUT, TEXTAREA...
prep_url	Thêm chuỗi http:// vào URL nếu không có
strip_image_tags	Lọc lấy địa chỉ URL của hình trong tag IMG
encode_php_tags	Chuyển tag của PHP thành các thực thể ký tự

Thư viện Form Validation còn cho phép thiết lập điều kiện kiểm tra bằng cách truyền vào hàm set_rules() một mảng hai chiều có dạng:

```
$config = array(
                array(
                      'field'   => 'txtUsername',
                      'label'   => 'Username',
                      'rules'   => 'required'
                   ),
                array(
                      'field'   => 'txtPassword',
                      'label'   => 'Password',
                      'rules'   => 'required'
                   ),
                array(
                      'field'   => 'txtPassConf',
                      'label'   => 'Password Confirmation',
                      'rules'   => 'required|matches[txtPassword]'
                   ),
                array(
                      'field'   => 'txtEmail',
                      'label'   => 'Email',
                      'rules'   => 'required'
                   )
            );

$this->form_validation->set_rules($config);
```

Ta cũng có thể sử dụng các hàm một đối số của PHP như trim(), htmlspecialchars(), md5()...để thiết lập điều kiện. Ngoài ra, CodeIgniter còn cho phép lập trình viên sử dụng các hàm callback tự định nghĩa để thiết lập luật kiểm tra của riêng mình, chẳng hạn như kiểm tra tên đăng nhập đã có trong cơ sở dữ liệu chưa...Các hàm callback được bắt đầu bằng tiền tố callback_ và phải trả về giá trị boolean. Ví dụ:

```
$this->form_validation->set_rules('txtUsername', 'Username',
'callback_check_username');

function check_username($str)
{
    if( $this->UserModel->usernamExists() == TRUE)
        return FALSE;
    else
        return TRUE;
}
```

Sau khi thiết lập các điều kiện, ta sử dụng phương thức `run()` để kiểm tra dữ liệu đầu vào. Nếu kết quả trả về là `TRUE`, tức là dữ liệu thỏa mãn các điều kiện đưa ra, ngược lại hàm trả về `FALSE`.

```
$this->form_validation->run();
```

Một chức năng hữu ích khác của thư viện CodeIgniter là cho phép ta lưu các điều kiện kiểm tra dữ liệu vào một tập tin thiết lập. Các điều kiện này có thể được sắp xếp vào từng nhóm. Các nhóm này có thể được gọi một cách tự động hay thông qua lời gọi của lập trình viên.

Để sử dụng chức năng này, ta tạo một tập tin `form_validation.php` trong thư mục `application/config`. Bên trong tập tin này, ta khai báo mảng `$config` chứa các điều kiện kiểm tra như đã trình bày, với tên của khóa cũng là tên nhóm. Ví dụ, tạo hai nhóm điều kiện có tên là `signup` và `email`:

```
$config = array(
                'signup' => array(
                                array(
                                        'field' => 'username',
                                        'label' => 'Username',
                                        'rules' => 'required'
                                     ),
                                array(
                                        'field' => 'password',
                                        'label' => 'Password',
                                        'rules' => 'required'
                                     ),
                'email' => array(
                                array(
                                        'field' => 'name',
                                        'label' => 'Name',
                                        'rules' => 'required|alpha'
                                     ),
                                array(
                                        'field' => 'title',
                                        'label' => 'Title',
                                        'rules' => 'required'
                                     ),
                                array(
                                        'field' => 'message',
                                        'label' => 'MessageBody',
```

```
                                        'rules' => 'required'
                                    )
                                )
                );
```

Khi đó, để kiểm tra dữ liệu cho một nhóm, ta gọi phương thức `run()` với tham số truyền vào là tên của nhóm điều kiện:

```
$this->form_validation->run('group_name');
```

Lớp Form Validation còn cho phép gắn nhóm điều kiện với từng phương thức trong controller. Khi dữ liệu được gửi vào phương thức này, lớp Form Validation sẽ tự động kiểm tra các điều kiện đã được định sẵn. Ví dụ, để thiết lập điều kiện kiểm tra dữ liệu cho phương thức `signup()` của lớp controller `Member`, ta khai báo:

```
$config = array(
            'member/signup' => array(
                                    array(
                                            'field' => 'username',
                                            'label' => 'Username',
                                            'rules' => 'required'
                                        ),
                                    array(
                                            'field' => 'password',
                                            'label' => 'Password',
                                            'rules' => 'required'
                                        ),
                                    array(
                                            'field' => 'passconf',
                                            'label' => 'PasswordConfirmation',
                                            'rules' => 'required'
                                        ),
                                    array(
                                            'field' => 'email',
                                            'label' => 'Email',
                                            'rules' => 'required'
                                        )
                                    )
                );
```

3.2.2. Xử lý lỗi

Thư viện Form Validation cung cấp hàm `validation_errors()` để hiển thị tất cả lỗi kiểm tra dữ liệu trong các tập tin view. Các thông báo sẽ hiển thị được quy định trong tập tin `system/language/english/form_validation_lang.php`. Ta cũng có thể thay đổi các thông báo này bằng cách sử dụng hàm `set_message()`.

```
$this->form_validation->set_message('rule', 'Error Message');
```

Với *rule* là tên điều kiện dựng sẵn của thư viện Form Validation và *Error Message* là thông báo sẽ được hiển thị. Nếu trong *Error Message* có chứa %s, giá trị label sẽ được thay thế vào đó.

```
$this->form_validation->set_message(required', 'Field %s must not be empty!');
```

Để thiết lập thông báo cho các hàm callback, ta chỉ cần đưa tên hàm làm đối số đầu tiên. Chẳng hạn:

```
$this->form_validation->set_message('check_username', 'This username is used.
Please choose another one!');
```

Để hiển thị lỗi cho riêng từng trường, ta có thể sử dụng hàm form_error() như sau:

```
<?php echo form_error($fieldName); ?> // Với $fieldName là tên của trường
```

Theo mặc định, các thông báo lỗi sẽ được đặt trong tag P. Ta có thể thiết lập lại cách hiển thị này cho phù hợp với giao diện website. CodeIgniter cho phép thiết lập toàn cục, áp dụng với tất cả các lần gọi hàm validation_errors(), và cục bộ, áp dụng với từng lần gọi hàm riêng biệt. Để thiết lập toàn cục, ta sử dụng hàm sau:

```
$this->form_validation->set_error_delimiters(string $startTag, string $endTag);
```

Chẳng hạn, đoạn mã dưới đây sẽ thiết lập các thông báo lỗi được đặt trong tag DIV.

```
$this->form_validation->set_error_delimiters('<div class="error">', '</div>');
```

Để thiết lập riêng cho từng lần gọi hàm, ta sử dụng:

```
<?php echo form_error('field name', '<div class="error">', '</div>'); ?>
```

Hoặc

```
<?php echo validation_errors('<div class="error">', '</div>'); ?>
```

3.2.3. Các hàm tiện ích

```
form_error(string $fieldName[, string $errorStartTag[, string $errorEndTag]])
```

Hàm form_error() sẽ hiển thị lỗi của trường được truyền vào.

```
set_value(string $fieldName[, mixed $defaultValue])
```

Hàm set_value() sẽ hiển thị lại những dữ liệu do người dùng nhập vào, trong trường hợp trường hợp xảy ra lỗi. Hàm này được áp dụng với các textfield (tag INPUT) hay textarea (tag TEXTAREA). Hàm nhận tên trường làm đối số thứ nhất. Đối số thứ hai (tùy chọn) sẽ hiển thị giá trị mặc định của trường khi được tải lần đầu.

```
set_select(string $fieldName[, mixed $defaultValue[, boolean $isSelected]])
```

Hàm `set_select()` sẽ hiển thị lại giá trị đã chọn của người dùng trong combo box, trong trường hợp xảy ra lỗi. Hàm được áp dụng cho các tag `OPTION`. Hàm nhận tên trường làm đối số thứ nhất. Đối số thứ hai (tùy chọn) sẽ hiển thị giá trị mặc định của tùy chọn này. Đối số thứ ba (tùy chọn) sẽ đánh dấu lựa chọn này làm lựa chọn mặc định.

```
set_checkbox(string $fieldName, mixed $defaultValue[, boolean $isSelected])
```

Hàm `set_checbox()` sẽ chọn những checkbox đã chọn của người dùng, trong trường hợp xảy ra lỗi. Hàm được áp dụng cho các tag `INPUT` với `type="checkbox"`. Hàm nhận tên trường làm đối số thứ nhất. Đối số thứ hai là giá trị của checkbox. Đối số thứ ba (tùy chọn) sẽ đánh dấu checkbox này được chọn.

```
set_radio(string $fieldName, mixed $defaultValue[, boolean $isSelected])
```

Hàm `set_radio()` sẽ chọn những radio button đã được chọn của người dùng, trong trường hợp xảy ra lỗi. Hàm này được áp dụng cho những tag `INPUT` có `type="radio"`. Hàm nhận tên trường làm đối số thứ nhất. Đối số thứ hai là giá trị của checkbox. Đối số thứ ba (tùy chọn) sẽ đánh dấu chọn cho radio button này.

3.3. Database

Thư viện Database là một thư viện quan trọng trong CodeIgniter. Thư viện này giúp cho lập trình viên thực hiện các thao tác với cơ sở dữ liệu, theo hai hướng tiếp cận: thủ tục truyền thống và Active Record. Để sử dụng thư viện Database, ta sử dụng đoạn mã sau:

```
$this->load->database();
```

Sau khi khai báo sử dụng thư viện, ta có thể truy xuất đến các phương thức của thư viện bằng đối tượng `$this->db`.

3.3.1. Thiết lập thông tin cơ sở dữ liệu

Thông tin cơ sở dữ liệu của hệ thống được lưu trong tập tin `application/config/database.php`. Các thông tin này được lưu trong một mảng hai chiều `$db` theo mẫu:

```
$db['default']['hostname'] = "localhost";
$db['default']['username'] = "root";
$db['default']['password'] = "";
$db['default']['database'] = "database_name";
$db['default']['dbdriver'] = "mysql";
$db['default']['dbprefix'] = "";
$db['default']['pconnect'] = TRUE;
```

```
$db['default']['db_debug'] = FALSE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = "";
$db['default']['char_set'] = "utf8";
$db['default']['dbcollat'] = "utf8_general_ci";
```

Trong đó

Khóa	Mô tả
hostname	Tên của máy chủ chứa cơ sở dữ liệu, ví dụ: localhost
username	Tên tài khoản truy cập
password	Mật khẩu truy cập
database	Tên cơ sở dữ liệu cần kết nối
dbdriver	Loại cơ sở dữ liệu, chẳng hạn: mysql, postgres, odbc…
dbprefix	Tiếp đầu ngữ của cơ sở dữ liệu, chẳng hạn như tbl_
pconnect	Giá trị boolean cho biết có sử dụng kết nối bền (persistent connection) hay không. Kết nối bền giúp cho hệ thống luôn chỉ mở một kết nối duy nhất đến cơ sở dữ liệu.
db_debug	Giá trị boolean cho biết có hiển thị lỗi của cơ sở dữ liệu hay không
cache_on	Giá trị boolean cho biết các truy vấn có được lưu trong bộ đệm hay không
cache_dir	Đường dẫn tuyệt đối đến thư mục đệm để lưu các truy vấn
char_set	Character set được sử dụng để giao tiếp với cơ sở dữ liệu
dbcollat	Character collation được sử dụng để giao tiếp với cơ sở dữ liệu
port	Cổng kết nối, sử dụng trong trường hợp kết nối đến cơ sở dữ liệu Postgres SQL.

Các thông tin trên được thiết lập tùy thuộc vào loại cơ sở dữ liệu cần kết nối. chẳng hạn như nếu sử dụng SQLite, ta không cần `username` và `password`, và `database` sẽ là đường dẫn đến tập tin cơ sở dữ liệu.

Bằng cách sử dụng mảng hai chiều để lưu thông tin, CodeIgniter cho phép ta thiết lập nhiều cơ sở dữ liệu trong cùng một ứng dụng (mỗi cơ sở dữ liệu sẽ được gọi là một nhóm). Khi cần kết nối đến cơ sở dữ liệu nào, ta chỉ cần sử dụng các thông số của cơ sở

dữ liệu đó. CodeIgniter sẽ chọn cơ sở dữ liệu mặc định bằng giá trị của biến `$active_group`.

3.3.2. Kết nối đến cơ sở dữ liệu

Ta có thể cho CodeIgniter tự động kết nối đến cơ sở dữ liệu mỗi khi hệ thống khởi động bằng cách thêm thư viện Database vào tập tin Autoloading. Cách làm này có thể gây lãng phí tài nguyên hệ thống vì đôi khi một số trang có thông tin tĩnh không cần đến xử lý cơ sở dữ liệu. CodeIgniter cho phép ta thực hiện kết nối bằng tay. Khi đó, cơ sở dữ liệu sẽ chỉ được gọi khi cần thiết.

Ví dụ:

```
// Kết nối đến cơ sở dữ liệu mặc định
$this->load->database();

// Kết nối đến một nhóm được quy định trong tập tin cấu hình cơ sở dữ liệu
$this->load->database('groupName');

// Kết nối đến cơ sở dữ liệu bất kỳ
$config['hostname'] = "localhost";
$config['username'] = "myusername";
$config['password'] = "mypassword";
$config['database'] = "mydatabase";
$config['dbdriver'] = "mysql";
$config['dbprefix'] = "";
$config['pconnect'] = FALSE;
$config['db_debug'] = TRUE;
$config['cache_on'] = FALSE;
$config['cachedir'] = "";
$config['char_set'] = "utf8";
$config['dbcollat'] = "utf8_general_ci";
$this->load->database($config);

// Hoặc sử dụng Data Source Name
$dsn =
'dbdriver://username:password@hostname/database?char_set=utf8&dbcollat=utf8_gene
ral_ci&cache_on=true&cachedir=/path/to/cache';

$this->load->database($dsn);
```

Trong các ứng dụng phức tạp, ta thường thao tác với nhiều cơ sở dữ liệu cùng lúc. Thư viện Database cho phép thực hiện yêu cầu này, bằng cách truyền giá trị TRUE làm đối số thứ hai khi gọi đến phương thức kết nối:

```
$fDb = $this->load->database('group_one', TRUE);
$sDb = $this->load->database('group_two', TRUE);
```

3.3.3. Truy vấn dữ liệu

Để thực hiện một câu truy vấn nào đó, ta có thể sử dụng phương thức sau:

```
$this->db->query(string $sql);
```

Phương thức `query()` trả về một đối tượng resource khi thực hiện câu lệnh `SELECT`, và trả về giá trị boolean khi thực hiện câu lệnh `INSERT`, `UPDATE`, `DELETE` cho biết truy vấn có được xử lý thành công không.

Truy vấn khi sử dụng phương thức `query()` sẽ không được tự động escape, điều này có thể làm xuất hiện lỗ hổng cho SQL Injection. Thư viện Database cung cấp 3 phương thức để escape dữ liệu trước khi đưa vào phương thức `query()`:

```
$this->db->escape();
```

Phương thức `escape()` sẽ tự động nhận dạng kiểu dữ liệu truyền vào. Nếu đó là dữ liệu kiểu chuỗi, phương thức `escape()` sẽ tự động thêm dấu nháy đơn vào chuỗi.

```
$this->db->escape_str();
```

Phương thức `escape_str()` sẽ tự động thêm dấu nháy đơn vào dữ liệu cần escape, bất kể đó là kiểu dữ liệu gì.

```
$this->db->escape_like_str();
```

Phương thức `escape_like_str()` sẽ escape các ký tự đặc biệt trong từ khóa tìm kiếm (như %, _) để đưa vào câu lệnh `LIKE`.

Một cách thức an toàn hơn để thực hiện truy vấn dữ liệu là sử dụng binding, tức là giá trị của biến sẽ được thay thế cho ký tự đại diện (wildcard). Thư viện Database trong CodeIgniter sử dụng dấu ? làm ký tự đại diện. Ví dụ:

```
$sql = "SELECT * FROM some_table WHERE id = ? AND status = ? AND author = ?";

$this->db->query($sql, array(3, 'live', 'Rick'));
```

Từng phần tử của mảng sẽ được lần lượt thay thế các dấu ? tương ứng. Dữ liệu này sẽ được tự động escape, giúp cho câu truy vấn trở nên an toàn hơn.

Sau khi thực hiện truy vấn, phương thức `query()` sẽ trả về một đối tượng resource chứa các kết quả. CodeIgniter cung cấp cho chúng ta một số phương thức để xử lý đối tượng này:

result()

Cú pháp

```
$query->result()
```

Phương thức `result()` giúp ta truy cập các kết quả trả về của truy vấn. Phương thức này chứa một mảng các đối tượng kết quả nếu truy vấn thành công, ngược lại trả về một mảng rỗng. Phương thức `result()` tương đương với phương thức `result_object()`.

Ví dụ:

```
$query = $this->db->query($sql);

foreach ($query->result() as $row)
{
   echo $row->title;
   echo $row->name;
   echo $row->body;
}
```

result_array()

Cú pháp

```
$query->result_array()
```

Phương thức `result_array()` có chức năng như phương thức `result()`, nhưng mỗi mẩu tin được lưu trong mảng. Kết quả trả về là một mảng các mẩu tin.

Ví dụ:

```
$query = $this->db->query($sql);

foreach ($query->result_array() as $row)
{
   echo $row['title'];
   echo $row['name'];
   echo $row['body'];
}
```

row()

Cú pháp

```
$query->row([int $index])
```

Phương thức `row()` chỉ trả về một mẩu tin duy nhất. Nếu kết quả có nhiều mẩu tin, mẩu tin đầu tiên sẽ được chọn. Kết quả trả về sẽ là một đối tượng.

```
$query = $this->db->query($sql);

if ($query->num_rows() > 0)
{
   $row = $query->row();

   echo $row->title;
   echo $row->name;
```

```
    echo $row->body;
}
```

Ta có thể chọn một mẩu tin cụ thể bằng cách truyền vào thứ tự của mẩu tin làm đối số thứ nhất.

```
$row = $query->row(3);
```

row_array()

Cú pháp

```
$query->row_array([int $index])
```

Phương thức `row_array()` có chức năng giống như phương thức `row()`, nhưng mẩu tin trả về sẽ là một mảng, thay vì một đối tượng.

Ví dụ:

```
$query = $this->db->query($sql);

if ($query->num_rows() > 0)
{
   $row = $query->row_array();

   echo $row['title'];
   echo $row['name'];
   echo $row['body'];
}
```

CodeIgniter còn cung cấp một số phương thức để di chuyển trong tập kết quả trả về, bao gồm:

```
// Lấy mẩu tin đầu tiên
$row = $query->first_row()

// Lấy mẩu tin cuối cùng
$row = $query->last_row()

// Lấy mẩu tin tiếp theo
$row = $query->next_row()

// Lấy mẩu tin phía trước
$row = $query->previous_row()
```

Theo mặc định, những phương thức này trả về một đối tượng kết quả. Để nhận được một mảng giá trị, ta truyền chuỗi `'array'` làm đối số thứ nhất.

```
$row = $query->first_row('array')
$row = $query->last_row('array')
$row = $query->next_row('array')
$row = $query->previous_row('array')
```

Một số phương thức hỗ trợ

num_rows()

Cú pháp

```
$query->num_rows()
```

Phương thức `num_rows()` trả về số mẩu tin trong một tập kết quả.

Ví dụ:

```
$query = $this->db->query("YOUR QUERY");

if ($query->num_rows() > 0)
{
   $row = $query->row();

   echo $row->title;
   echo $row->name;
   echo $row->body;
}
```

num_fields()

Cú pháp

```
$query->num_fields()
```

Phương thức `num_fields()` trả về số trường dữ liệu trong truy vấn.

Ví dụ:

```
$query = $this->db->query('SELECT * FROM my_table');

echo $query->num_fields();
```

free_result()

Cú pháp

```
$query->free_result()
```

Phương thức `free_result()` sẽ giải phóng vùng bộ nhớ đang chứa kết quả. Thông thường, PHP sẽ tự động giải phóng bộ nhớ sau khi thực hiện xong các lệnh. Trong một số trường hợp phải xử lý nhiều truy vấn, ta cần giải phóng bộ nhớ để nâng cao hiệu năng chương trình. Ví dụ:

```
$query = $this->db->query('SELECT title FROM my_table');

foreach ($query->result() as $row)
```

```
{
   echo $row->title;
}
$query->free_result();
// Biến $query đã được giải phóng

$query2 = $this->db->query('SELECT name FROM some_table');

$row = $query2->row();
echo $row->name;
$query2->free_result();
// Biến $query2 đã được giải phóng
```

3.3.4. Active Record

Trong công nghệ phần mềm, Active Record là một mẫu thiết kế được sử dụng trong các ứng dụng có liên quan đến cơ sở dữ liệu. Active Record xây dựng một giao diện (interface) có các phương thức như `select()`, `insert()`, `update()`, `delete()`...và thuộc tính ít nhiều tương ứng với các cột trong bảng dữ liệu.

Active Record là một hướng tiếp cận để truy xuất dữ liệu. Một bảng dữ liệu hay một khung nhìn (view) được ánh xạ thành một lớp. Khi đó, một đối tượng của lớp tương ứng với một dòng trong bảng. Nếu một đối tượng được cập nhật thông tin thì cơ sở dữ liệu cũng được cập nhật.

Active Record trong CodeIgniter được xây dựng nhằm hỗ trợ lập trình viên thực hiện các thao tác với cơ sở dữ liệu nhanh chóng, đơn giản và an toàn hơn. Các câu truy vấn được xây dựng một cách tự động, tùy thuộc vào loại cơ sở dữ liệu ta đang thao tác. Dữ liệu được đưa vào cũng không cần phải escape, Active Record sẽ tự động thực hiện việc này. Ví dụ:

```
$this->db->get('tblName');
// Tương đương với câu lệnh SQL:
// SELECT * FROM `tblName`

$this->db->select('username, password')
$this->db->from('user');
// Tương đương với câu lệnh SQL:
// SELECT `username`, `password` FROM `tbl_user`
```

Lưu ý: Nếu sử dụng PHP5, ta có thể liên kết các câu phương thức lại với nhau (method chaining) như trong ví dụ sau:

```
$this->db->select('username, password')->from('user');
```

Active Record được kích hoạt bằng cách thiết lập giá trị trong tập tin `application/config/database.php`.

```
$active_record = TRUE;
```

3.3.5. Truy vấn dữ liệu

Thư viện Database hỗ trợ xây dựng các câu lệnh `SELECT` bằng những phương thức sau:

get()

Cú pháp:

```
$this->db->get([string $tableName[, int $limit, int $offset]]);
```

Hàm `get()` sẽ trả về một đối tượng resource chứa kết quả của câu truy vấn. Nếu đối số thứ nhất được truyền vào, hàm sẽ lấy tất cả các dòng trong bảng dữ liệu.

```
$this->db->get('data');
// Tương đương với câu truy vấn: SELECT * FROM `tbl_data`
```

Đối số thứ hai và thứ ba cho phép giới hạn số kết quả trả về.

```
$this->db->get('data', 10, 20);
// Tương đương với câu truy vấn: SELECT * FROM `tbl_data` LIMIT 10, 20
```

Trong trường hợp không có đối số nào được truyền vào, hàm `get()` sẽ thực hiện truy vấn với câu lệnh được xây dựng bởi các phương thức `select()`, `from()`, `where()`...

Ví dụ:

```
$this->db->select('username, password');
$this->db->from('member');
$this->db->where(array('user_id' => 1));
$query = $this->db->get();
// Tương đương với truy vấn:
// SELECT `username`, `password` FROM `tbl_member` WHERE `user_id` = 1
```

get_where()

Cú pháp:

```
$this->db->get([string $tableName[, array $condtions, [, int $limit, int
$offset]]]);
```

Hàm `get_where()` tương tự như hàm `get()`. Điểm khác biệt nằm ở đối số thứ hai của hàm. Đối số này nhận một mảng làm giá trị, giúp xây dựng mệnh đề `WHERE`, thay vì phải sử dụng hàm `$this->db->where()`.

Ví dụ:

```
$query = $this->db->get_where('member', array('user_id' => 1));
// Tương đương với truy vấn:
// SELECT * FROM `tbl_member` WHERE `user_id` = 1
```

select()

Cú pháp

```
$this->db->select([string $fields = '*'[, boolean $isProtected = TRUE]]);
```

Hàm `select()` giúp ta chọn những trường cần thiết trong bảng dữ liệu bằng cách truyền vào chuỗi chứa tên trường làm đối số thứ nhất, các trường cách nhau bởi dấu phẩy (,). Nếu không có đối số truyền vào, mặc định sẽ lấy tất cả (tương đương với `SELECT *`...).

```
$this->db->select('title, content, date');
$query = $this->db->get('mytable');

// Tương đương: SELECT `title`, `content`, `date` FROM `mytable`
```

Đối số thứ hai cho biết CodeIgniter có bảo vệ tên trường và tên bảng trong truy vấn bằng ký tự backtick (`) hay không.

```
$this->db->select('(SELECT SUM(payments.amount) FROM payments WHERE
payments.invoice_id=4') AS amount_paid', FALSE);
$query = $this->db->get('mytable');
```

select_max()

Cú pháp:

```
$this->db->select_max(string $fieldName[, string $alias])
```

Phương thức `select_max()` cho phép ta sử dụng hàm `MAX()` trên một thuộc tính trong bảng dữ liệu. Đối số thứ hai (tùy chọn) cho phép ta tạo tên thay thế (alias) cho kết quả trả về.

Ví dụ:

```
$this->db->select_max('age', 'member_age');
$query = $this->db->get('members');
// Tương ứng: SELECT MAX(age) as member_age FROM members
```

select_min()

Cú pháp

```
$this->db->select_min (string $fieldName[, string $alias])
```

Tương tự như `select_max()`, phương thức `select_min()` cho phép ta sử dụng hàm `MIN()` trên một thuộc tính trong bảng dữ liệu. Đối số thứ hai (tùy chọn) cho phép tạo tên thay thế cho kết quả trả về.

Ví dụ:

```
$this->db->select_min('age');
$query = $this->db->get('members');
// Tương ứng: SELECT MIN(age) AS age FROM members
```

select_avg()

Cú pháp

```
$this->db->select_avg(string $fieldName[, string $alias])
```

Phương thức `select_avg()` cho phép ta sử dụng hàm `AVG()` trên một thuộc tính trong bảng dữ liệu. Đối số thứ hai (tùy chọn) cho phép tạo tên thay thế cho kết quả trả về.

Ví dụ:

```
$this->db->select_avg('age');
$query = $this->db->get('members');
// Tương ứng: SELECT AVG(age) as age FROM members
```

select_sum()

Cú pháp

```
$this->db->select_sum(string $fieldName[, string $alias])
```

Phương thức `select_sum()` cho phép ta sử dụng hàm `SUM()` trên một thuộc tính trong bảng dữ liệu. Đối số thứ hai (tùy chọn) cho phép tạo tên thay thế cho kết quả trả về.

Ví dụ:

```
$this->db->select_sum('age');
$query = $this->db->get('members');
// Tương ứng: SELECT SUM(age) AS age FROM members
```

from()

Cú pháp

```
$this->db->from(string $tableName)
```

Phương thức `from()` xây dựng câu lệnh `FROM`, cho phép ta chọn những bảng dữ liệu để truy vấn.

```
$this->db->select('title, content, date');
$this->db->from('mytable');

$query = $this->db->get();
```

```
// Tương ứng: SELECT title, content, date FROM mytable
```

Ta cũng có thể sử dụng phương thức `get()` để chọn bảng dữ liệu trong truy vấn.

join()

Cú pháp

```
$this->db->join(string $tableName, string $condition[, string $nature])
```

Phương thức `join()` xây dựng câu lệnh `JOIN`, cho phép kết hợp các bảng dữ liệu với nhau. Phương thức nhận tên bảng cần kết hợp làm đối số thứ nhất và điều kiện liên kết làm đối số thứ hai.

```
$this->db->select();
$this->db->from('blogs');
$this->db->join('comments', 'comments.id = blogs.id');

$query = $this->db->get();

// Tương ứng:
// SELECT * FROM blogs
// JOIN comments ON comments.id = blogs.id
```

Để có thể thực hiện phép kết với nhiều bảng dữ liệu, ta gọi phương thức `join()` nhiều lần. Đối số thứ ba (tùy chọn) trong phương thức cho phép ta thiết lập kiểu liên kết, bao gồm: left, right, outer, inner, left outer, and right outer.

```
$this->db->join('comments', 'comments.id = blogs.id', 'left');

// Tương ứng: LEFT JOIN comments ON comments.id = blogs.id
```

where()

Thư viện Database của CodeIgniter cung cấp 4 cách để xây dựng điều kiện `WHERE` cho truy vấn. Dữ liệu được truyền vào phương thức này sẽ được tự động escape, giúp cho câu truy vấn an toàn hơn.

1. *So sánh một thuộc tính*

Cú pháp

```
$this->db->where(string $field, string $value)
```

Phương thức nhận tên của thuộc tính làm đối số thứ nhất và giá trị của thuộc tính làm đối số thứ hai. Nếu phương thức được gọi nhiều lần, các điều kiện sẽ được liên kết với nhau bằng toán tử `AND`.

```
$this->db->where('name', $name);
```

```
// Tương ứng: WHERE name = 'Joe'

$this->db->where('name', $name);
$this->db->where('title', $title);
$this->db->where('status', $status);

// Tương ứng: WHERE name 'Joe' AND title = 'boss' AND status = 'active'
```

Ta có thể chèn các toán tử khác thay cho toán tử so sánh bằng (=) vào giá trị trong đối số thứ nhất.

```
$this->db->where('name !=', $name);
$this->db->where('id <', $id);

// Tương ứng: WHERE name != 'Joe' AND id < 45
```

2. Sử dụng mảng

Cú pháp

```
$this->db->where(array $conditions)
```

Ta có thể thiết lập nhiều điều kiện cùng lúc bằng cách sử dụng mảng một chiều làm đối số thứ nhất cho phương thức `where()`.

```
$array = array('name' => $name, 'title' => $title, 'status' => $status);
$this->db->where($array);

// Tương ứng: WHERE name = 'Joe' AND title = 'boss' AND status = 'active'
```

Cũng như cách 1, ta cũng có thể chèn các toán tử so sánh khác toán tử bằng.

```
$array = array('name !=' => $name, 'id <' => $id, 'date >' => $date);

$this->db->where($array);
```

3. Sử dụng chuỗi

```
$where = "name='Joe' AND status='boss' OR status='active'";
$this->db->where($where);
```

Phương thức `where()` chấp nhận một đối số thứ 3 (tùy chọn). Nếu đối số này được thiết lập bằng `TRUE`, CodeIgniter sẽ không bảo vệ tên các thuộc tính và bảng dữ liệu bằng ký tự backtick (`). Ví dụ:

```
$this->db->where('MATCH (field) AGAINST ("value")', NULL, FALSE);
```

or_where()

Phương thức `or_where()` giống như phương thức `where()` nhưng các điều kiện được liên kết với nhau bằng toán tử `OR`.

Ví dụ:

```
$this->db->where('name !=', $name);
$this->db->or_where('id >', $id);

// Tương ứng: WHERE name != 'Joe' OR id > 50
```

where_in()

Phương thức `where_in()` sẽ tạo các truy vấn dạng `WHERE…IN…` Nếu phương thức này được gọi nhiều lần, các điều kiện sẽ được liên kết với nhau bằng lệnh `AND`.

Ví dụ:

```
$names = array('Frank', 'Todd', 'James');
$this->db->where_in('username', $names);
// Tương ứng: WHERE username IN ('Frank', 'Todd', 'James')
```

or_where_in()

Phương thức `or_where_in()` có chức năng giống như phương thức `where_in()`, chỉ khác ở chỗ, các điều kiện sẽ được liên kết với nhau bằng lệnh `OR` nếu phương thức được gọi nhiều lần.

Ví dụ:

```
$names = array('Frank', 'Todd', 'James');
$this->db->or_where_in('username', $names);
// Tương ứng: OR username IN ('Frank', 'Todd', 'James')
```

where_not_in()

Phương thức `where_not_in()` sẽ tạo các truy vấn dạng `WHERE…NOT IN…`Nếu phương thức này được gọi nhiều lần, các điều kiện sẽ được liên kết với nhau bằng lệnh `AND`.

Ví dụ:

```
$names = array('Frank', 'Todd', 'James');
$this->db->where_not_in('username', $names);
// Produces: WHERE username NOT IN ('Frank', 'Todd', 'James')
```

or_where_not_in()

Phương thức `or_where_not_in()` có chức năng giống phương thức `where_not_in()`, nhưng các điều kiện sẽ được liên kết với nhau bằng lệnh `OR`.

Ví dụ:

```
$names = array('Frank', 'Todd', 'James');
$this->db->or_where_not_in('username', $names);
```

```
// Produces: OR username NOT IN ('Frank', 'Todd', 'James')
```

like()

Phương thức `like()` giúp xây dựng câu lệnh `LIKE` trong truy vấn. CodeIgniter cho phép ta sử dụng phương thức này theo hai cách:

1. *Sử dụng giá trị*

Cú pháp

```
$this->db->like(string $field, mixed $value[, string $wildcardPlace]);
```

Phương thức `like()` nhận tên trường cần so sánh làm đối số thứ nhất và giá trị cần so sánh làm đối số thứ hai. Đối số thứ ba (tùy chọn) cho phép ta đặt vị trí của ký tự % trong giá trị so sánh. Đối số này có 3 giá trị: before (dấu % sẽ được đặt trước giá trị), after (dấu % sẽ được đặt sau giá trị) và both (dấu % sẽ được đặt ở cả hai đầu). Nếu phương thức này được gọi nhiều lần, các điều kiện sẽ được liên kết với nhau bằng lệnh `AND`.

Ví dụ:

```
$this->db->like('title', 'match');
$this->db->like('body', 'match', 'before');


// Tương ứng: WHERE title LIKE '%match%' AND body LIKE '%match'
```

2. *Sử dụng mảng*

Ta cũng có thể sử dụng mảng một chiều để xây dựng phương thức. Mảng này có khóa là tên của trường cần so sánh và giá trị là từ khóa cần tìm.

Ví dụ:

```
$array = array('title' => $match, 'page1' => $match, 'page2' => $match);

$this->db->like($array);

// Tương ứng: WHERE title LIKE '%match%' AND page1 LIKE '%match%' AND page2 LIKE
'%match%'
```

or_like()

Phương thức `or_like()` giống như phương thức `like()`, nhưng các điều kiện sẽ được liên kết với nhau bằng lệnh `OR`.

Ví dụ:

```
$this->db->like('title', 'match');
$this->db->or_like('body', $match);

// Tương ứng: WHERE title LIKE '%match%' OR body LIKE '%match%'
```

not_like()

Phương thức `not_like()` giúp xây dựng câu lệnh `NOT LIKE` trong truy vấn. Khi phương thức này được gọi nhiều lần, các điều kiện sẽ được liên kết với nhau bằng lệnh `AND`. Cách sử dụng phương thức này giống như phương thức `like()`.

Ví dụ:

```
$this->db->not_like('title', 'match');

// WHERE title NOT LIKE '%match%
```

or_not_like()

Phương thức `or_not_like()` cũng giống như phương thức `not_like()`, nhưng các điều kiện sẽ được liên kết với nhau bằng lệnh `OR`.

Ví dụ:

```
$this->db->like('title', 'match');
$this->db->or_not_like('body', 'match');

// Tương ứng: WHERE title LIKE '%match% OR body NOT LIKE '%match%'
```

group_by()

Cho phép xây dựng câu lệnh `GROUP BY`. Phương thức này nhận tên của trường cần nhóm lại làm đối số thứ nhất. Trong trường hợp cần nhóm nhiều trường, ta có thể truyền vào một mảng chứa tên các trường.

Ví dụ:

```
$this->db->group_by('title');
// Tương ứng: GROUP BY title

$this->db->group_by(array('title', 'date'));
// Tương ứng: GROUP BY title, date
```

having()

Phương thức `having()` cho phép thêm các điều kiện để chọn nhóm trong truy vấn. Phương thức này nhận tên trường làm đối số thứ nhất, điều kiện chọn làm đối số thứ 2. Hoặc ta có thể truyền một mảng chứa các điều kiện vào làm đối số thứ nhất.

Ví dụ:

```
$this->db->having('user_id = 45');
// Tương ứng: HAVING user_id = 45

$this->db->having('user_id', 45);
// Tương ứng: HAVING user_id = 45

$this->db->having(array('title =' => 'My Title', 'id <' => $id));
// Tương ứng: HAVING title = 'My Title', id < 45
```

CodeIgniter tự động bảo vệ tên các trường và bảng dữ liệu bằng ký tự backtick (`). Nếu không muốn thực hiện đều này, ta truyền giá trị `FALSE` vào làm đối số thứ ba của phương thức.

Ví dụ:

```
$this->db->having('user_id', 45);
// Tương ứng: HAVING `user_id` = 45

$this->db->having('user_id', 45, FALSE);
// Tương ứng: HAVING user_id = 45
```

or_having()

Phương thức này giống như phương thức `having()`, nhưng các điều kiện sẽ được liên kết với nhau bằng lệnh `OR`.

distint()

Phương thức `distint()` sẽ thêm từ khóa `DISTINT` vào câu truy vấn.

Ví dụ:

```
$this->db->distinct();
$this->db->get('table');

// Tương ứng: SELECT DISTINCT * FROM table
```

order_by()

Phương thức `order_by()` giúp xây dựng câu lệnh `ORDER BY` trong truy vấn, nhằm sắp xếp các giá trị trả về. Phương thức này nhận tên trường cần sắp xếp làm đối số thứ nhất. Đối số thứ hai sẽ quyết định cách thức sắp xếp, có 3 giá trị mặc định: asc (từ trên xuống – mặc định), desc (từ dưới lên) và random (ngẫu nhiên).

Ví dụ:

```
$this->db->order_by("title", "desc");
// Tương ứng: ORDER BY title DESC
```

```
$this->db->order_by('title desc, name asc');
// Tương ứng: ORDER BY title DESC, name ASC

$this->db->order_by("title", "desc");
$this->db->order_by("name", "asc");
// Tương ứng: ORDER BY title DESC, name ASC
```

limit()

Phương thức `limit()` giúp xây dựng câu lệnh `LIMIT`, nhằm giới hạn số lượng kết quả trả về. Phương thức này nhận số kết quả trả về làm đối số thứ nhất, đối số thứ hai thể hiện cột mốc để lấy kết quả.

Ví dụ:

```
$this->db->limit(10);
// Tương ứng: LIMIT 10

$this->db->limit(10, 20);
// Tương ứng: LIMIT 20, 10
```

count_all_results()

Phương thức `count_all_results()` trả về số kết quả của một câu truy vấn.

Ví dụ:

```
$this->db->like('title', 'match');
$this->db->from('my_table');
echo $this->db->count_all_results();
// Trả về một số nguyên, ví dụ 17
```

count_all()

Phương thức `count_all()` trả về số mẩu tin trong một bảng dữ liệu. Phương thức này nhận tên bảng làm đối số thứ nhất.

Ví dụ:

```
echo $this->db->count_all('my_table');
// Trả về số mẩu tin trong bảng dữ liệu my_table
```

3.3.6. Thao tác dữ liệu

Để thực hiện các thao tác thêm, xóa, sửa dữ liệu, thư viện Database trong CodeIgniter cung cấp cho những phương thức sau:

insert()

Cú pháp

```
$this->db->insert(string $table, array/object $values);
```

Phương thức `insert()` giúp chèn một dòng mới vào bảng dữ liệu. Phương thức này nhận tên của bảng làm đối số thứ nhất. Đối số thứ hai là một mảng có khóa là tên trường và giá trị của khóa cũng là giá trị cần chèn vào.

Ví dụ:

```
$data = array(
                'title' => 'My title' ,
                'name' => 'My Name' ,
                'date' => 'My date'
            );

$this->db->insert('mytable', $data);

// Tương ứng: INSERT INTO mytable (title, name, date) VALUES ('My title', 'My
name', 'My date')
```

Ta cũng có thể truyền vào một đối tượng làm đối số thứ hai. Trong đó, các thuộc tính của đối tượng là tên trường, và giá trị của các thuộc tính này cũng là giá trị cần thêm vào.

Ví dụ:

```
/*
    class Myclass {
        var $title = 'My Title';
        var $content = 'My Content';
        var $date = 'My Date';
    }
*/

$object = new Myclass;

$this->db->insert('mytable', $object);

// Tương ứng: INSERT INTO mytable (title, content, date) VALUES ('My Title', 'My
Content', 'My Date')
```

Tất cả giá trị sẽ được escape, giúp cho truy vấn an toàn hơn.

update()

Cú pháp

```
$this->db->update(string $table, array/object $data[, array $conditions])
```

Phương thức `update()` giúp cập nhật dữ liệu trong bảng. Phương thức này nhận tên bảng làm đối số thứ nhất, các dữ liệu cần cập nhật làm đối số thứ hai, đối số thứ ba (tùy chọn) cho phép thiết lập điều kiện để cập nhật dữ liệu.

Ví dụ:

```
$data = array(
    'product_name' => 'Nike AirMax+ 2009',
    'product_price' => 160
);

$this->db->update('product', $data, array('product_id' => $id));
// Tương ứng
// UPDATE `tbl_product` SET `product_name` = 'Nike AirMax+ 2009',
// `product_price` = 160 WHERE `product_id` = {$id}
```

set()

Cú pháp:

```
$this->db->set(string $field, string $value[, boolean $isEscaped]);
```

Thay vì truyền vào một mảng hay đối tượng để chứa dữ liệu khi thực hiện thêm/cập nhật dữ liệu, ta có thể sử dụng phương thức `set()` để gán giá trị cho một trường cụ thể. Phương thức này nhận tên trường làm đối số thứ nhất, giá trị của trường làm đối số thứ hai.

Ví dụ:

```
$this->db->set('name', $name);
$this->db->set('title', $title);
$this->db->set('status', $status);
$this->db->insert('mytable');
// Tương ứng: INSERT INTO `tbl_mytable` (name, title, status) VALUES ($name,
$title, $status)
```

Đối số thứ ba (tùy chọn) cho biết dữ liệu của trường có được escape hay không. Mặc định là `TRUE`, ta có thể thay đổi bằng cách truyền vào giá trị `FALSE`.

Ví dụ:

```
$this->db->set('field', 'field+1', FALSE);
$this->db->insert('mytable');
// Tương ứng INSERT INTO mytable (field) VALUES (field+1)

$this->db->set('field', 'field+1');
$this->db->insert('mytable');
// Tương ứng INSERT INTO mytable (field) VALUES ('field+1')
```

Ta cũng có thể truyền vào một mảng/đối tượng để thiết lập giá trị cho nhiều trường cùng lúc.

Ví dụ:

```
$array = array('name' => $name, 'title' => $title, 'status' => $status);

$this->db->set($array);
$this->db->insert('mytable');

/*
    class Myclass {
        var $title = 'My Title';
        var $content = 'My Content';
        var $date = 'My Date';
    }
*/

$object = new Myclass;

$this->db->set($object);
$this->db->insert('mytable');
```

delete()

Cú pháp

```
$this->db->delete(string $table, array $conditions)
```

Phương thức `delete()` giúp xóa dữ liệu trong bảng. Phương thức này nhận tên bảng làm đối số thứ nhất. Đối số thứ hai giúp xác định điều kiện những mẩu tin nào sẽ được xóa.

Ví dụ:

```
$this->db->delete('mytable', array('id' => $id));

// Tương ứng: DELETE FROM `mytable` WHERE `id` = $id
```

Nếu một mảng các tên bảng được truyền vào, phương thức này sẽ thực hiện xóa dữ liệu trên nhiều bảng.

Ví dụ:

```
$tables = array('table1', 'table2', 'table3');
$this->db->where('id', '5');
$this->db->delete($tables);
```

empty_table()

Cú pháp

```
$this->db->empty_table(string $table)
```

Phương thức `empty_table()` giúp xóa tất cả mẩu tin trong một bảng dữ liệu bằng cách sử dụng câu lệnh `DELETE FROM`. Phương thức này nhận tên bảng làm đối số duy nhất.

Ví dụ:

```
$this->db->empty_table('mytable');
// Tương ứng: DELETE FROM mytable
```

truncate()

Cú pháp

```
$this->db->truncate(string $table)
```

Phương thức `truncate()` cũng giúp xóa tất cả mẩu tin trong bảng dữ liệu. Điểm khác biệt với phương thức `empty_table()` là `truncate()` không thể được thực hiện nếu bảng cần xóa làm khóa ngoại ở bảng khác. Bảng sau khi được xóa sẽ khởi động lại chỉ mục. Phương thức này cũng nhận tên bảng làm đối số duy nhất. Trong trường hợp câu lệnh `TRUNCATE` không được phép thực hiện, phương thức này sẽ sử dụng `DELETE FROM`.

Ví dụ:

```
$this->db->from('mytable');
$this->db->truncate();
// hoặc
$this->db->truncate('mytable');

// Tương ứng:
// TRUNCATE mytable
```

3.3.7. Lưu trữ truy vấn trong Active Record

Thông thường, sau khi thực hiện một truy vấn, thư viện Database sẽ thiết lập lại giá trị các biến cho lần truy vấn tiếp theo. Bằng cách lưu lại các thông tin này (caching), lập trình viên có thể ngăn chặn việc tái thiết lập giá trị các biến và sử dụng lại chúng một cách dễ dàng. CodeIgniter hỗ trợ caching cho các câu lệnh sau: `SELECT`, `FROM`, `JOIN`, `WHERE`, `LIKE`, `GROUP_BY`, `HAVING`, `ORDER_BY`, `SET`. Dưới đây là các phương thức hỗ trợ lưu trữ truy vấn:

start_cache()

Phương thức này sẽ được gọi khi bắt đầu caching. Tất cả các câu lệnh trong danh sách hỗ trợ sẽ được lưu lại.

stop_cache()

Phương thức này sẽ ngưng việc lưu trữ các truy vấn lại.

flush_cache()

Tất cả truy vấn đang được lưu trữ sẽ bị xóa.

Ví dụ:

```
$this->db->start_cache();
$this->db->select('field1');
$this->db->stop_cache();

$this->db->get('tablename');

// Tương ứng: SELECT `field1` FROM (`tablename`)

$this->db->select('field2');
$this->db->get('tablename');

// Tương ứng: SELECT `field1`, `field2` FROM (`tablename`)

$this->db->flush_cache();

$this->db->select('field2');
$this->db->get('tablename');

// Tương ứng: SELECT `field2` FROM (`tablename`)
```

3.3.8. Giao dịch (transaction) trong CodeIgniter

Thư viện Database của CodeIgniter cho phép thực hiện giao dịch (transaction) trên các hệ quản trị cơ sở dữ liệu có hỗ trợ. Đối với hệ quản trị MySQL, ta cần chọn kiểu lưu trữ bảng là InnoDB hoặc BDB thay vì kiểu MyISAM thông dụng để có thể thực hiện giao dịch.

Cách tiếp cận của CodeIgniter khi thực hiện giao dịch tương đối giống với thư viện ADODB. Bằng cách đó, lập trình viên có thể đơn giản hóa việc lập trình. Trong hầu hết các trường hợp, chỉ cần hai dòng lệnh là có thể thực hiện giao dịch.

Về căn bản, thực hiện giao dịch sẽ cần khá nhiều tài nguyên để cài đặt vì nó sẽ theo dõi các truy vấn. Nếu truy vấn thành công, giao dịch sẽ xác nhận (commit). Ngược lại, tất cả những truy vấn trước đó sẽ bị hủy, tình trạng của cơ sở dữ liệu sẽ được quay về trạng thái ban đầu (rollback).

Mặc định, thư viện Database thực hiện các giao dịch theo chế độ nghiêm ngặt (strict mode). Nếu thực hiện nhiều nhóm giao dịch, và xảy ra một nhóm bị lỗi, thì tất cả các nhóm khác sẽ được rollback. Trong trường hợp chế độ nghiêm ngặt bị vô hiệu hóa, các nhóm giao dịch sẽ được thực hiện độc lập với nhau. Để bật/tắt chế độ này, ta sử dụng hàm `trans_strict()` như sau:

```
$this->db->trans_strict(FALSE);
```

Để thực hiện giao dịch, ta sử dụng hai phương thức `trans_start()` và `trans_complete()` như sau:

```
$this->db->trans_start();
$this->db->query('AN SQL QUERY...');
$this->db->query('ANOTHER QUERY...');
$this->db->query('AND YET ANOTHER QUERY...');
$this->db->trans_complete();
```

Số truy vấn bên trong thân của hai phương thức này là không giới hạn. Hệ thống sẽ tự động commit hay rollback dựa vào kết quả của câu truy vấn.

Ngay sau khi khai báo phương thức `trans_start()`, hệ thống sẽ tự động thực hiện giao dịch. Nếu không muốn thực hiện giao dịch, ta có thể sử dụng phương thức `trans_off()`. Khi đó các câu truy vấn bên trong cặp hàm `trans_start()` và `trans_complete()` vẫn được thực hiện mà không có hỗ trợ giao dịch.

Thư viện Database còn cho phép thực hiện giao dịch theo chế độ kiểm tra (test mode). Nghĩa là sau khi thực hiện giao dịch, hệ thống sẽ được rollback bất kể các truy vấn thành công hay không. Để bật chế độ kiểm tra, ta truyền giá trị `TRUE` vào hàm `trans_start()` như sau:

```
$this->db->trans_start(TRUE); // Query will be rolled back
$this->db->query('AN SQL QUERY...');
$this->db->trans_complete();
```

Theo mặc định, nếu xảy ra lỗi khi thực hiện giao dịch, hệ thống sẽ tự động rollback. Ta có thể điều khiển quá trình này bằng cách sử dụng phương thức trans_begin(). Phương thức này thông báo cho CodeIgniter thực hiện giao dịch bằng tay (manually).

```
$this->db->trans_begin();

$this->db->query('AN SQL QUERY...');
$this->db->query('ANOTHER QUERY...');
$this->db->query('AND YET ANOTHER QUERY...');

if ($this->db->trans_status() === FALSE)
{
    $this->db->trans_rollback();
}
else
{
    $this->db->trans_commit();
}
```

3.3.9. Một số phương thức trợ giúp

insert_id()

Cú pháp:

```
$this->db->insert_id()
```

Phương thức này trả về số ID khi chèn một dòng mới vào cơ sở dữ liệu.

affected_rows()

Cú pháp

```
$this->db->affected_rows()
```

Phương thức `affected_rows()` trả về số dòng dữ liệu được tác động khi thực hiện các câu lệnh `INSERT`, `UPDATE`, `DELETE`. Trong MySQL, khi thực hiện truy vấn `DELETE FROM TABLE`, số dòng trả về luôn bằng 0. Tuy nhiên, thư viện Database trong CodeIgniter có thể trả về đúng số dòng đã được xóa.

count_all()

Cú pháp

```
$this->db->count_all(string $tableName)
```

Phương thức `count_all()` trả về số dòng dữ liệu đang có của một bảng.

platform()

Cú pháp

```
$this->db->platform()
```

Phương thức `platform()` trả về tên hệ quản trị cơ sở dữ liệu đang được sử dụng (MySQL, MSSQL, Postgres SQL…).

version()

Cú pháp

```
$this->db->version()
```

Phương thức `version()` trả về phiên bản hệ quản trị cơ sở dữ liệu đang sử dụng.

last_query()

Cú pháp

```
$this->db->last_query()
```

Phương thức `last_query()` trả về câu truy vấn vừa được thực hiện.

insert_string()

Cú pháp

```
$this->db->insert_string(string $tableName, array $data)
```

Phương thức `insert_string()` sẽ tạo ra câu lệnh `INSERT`. Phương thức này nhận tên bảng sẽ thêm dữ liệu làm đối số thứ nhất, và mảng chứa dữ liệu cần thêm làm đối số thứ hai. Ví dụ:

```
$data = array('name' => $name, 'email' => $email, 'url' => $url);
$str = $this->db->insert_string('table_name', $data);

// Kết quả
INSERT INTO table_name (name, email, url) VALUES ('Rick', 'rick@example.com',
'example.com')
```

update_string()

Cú pháp

```
$this->db->update_string(string $tableName, array $data, array $where)
```

Phương thức `insert_string()` sẽ tạo ra câu lệnh `UPDATE`. Phương thức này nhận tên bảng sẽ thêm dữ liệu làm đối số thứ nhất, mảng chứa dữ liệu cần thêm làm đối số thứ hai và mảng chứa các điều kiện cập nhật làm đối số thứ ba. Ví dụ:

```
$data = array('name' => $name, 'email' => $email, 'url' => $url);
$where = "author_id = 1 AND status = 'active'";

$str = $this->db->update_string('table_name', $data, $where);

// Kết quả
UPDATE table_name SET name = 'Rick', email = 'rick@example.com', url =
'example.com' WHERE author_id = 1 AND status = 'active'
```

Hai phương thức `insert_string()` và `update_string()` được dùng để tạo ra chuỗi truy vấn, giúp tạo câu lệnh nhanh chóng. Dữ liệu được truyền vào hai phương thức này đều được escape, giúp cho truy vấn an toàn hơn.

list_tables()

Cú pháp

```
$this->db->list_tables()
```

Phương thức `list_tables()` trả về một mảng chứa các tên các bảng của cơ sở dữ liệu đang kết nối.

table_exists()

Cú pháp

```
$this->db->table_exits(string $tableName)
```

Phương thức `table_exists()` giúp kiểm tra một bảng có tồn tại trong cơ sở dữ liệu đang kết nối hay không.

list_fields()

Cú pháp

```
$this->db->list_fields(string $tableName)
```

Phương thức `list_fields()` trả về một mảng chứa tên các trường của bảng truyền vào. Ngoài ra, phương thức này cũng có thể được sử dụng để lấy các trường được trả về trong một truy vấn, chẳng hạn:

```
$query = $this->db->query('SELECT * FROM some_table');

foreach ($query->list_fields() as $field)
{
   echo $field;
}
```

field_exists()

Cú pháp

```
$this->db->field_exits(string $fieldName, string $tableName)
```

Phương thức `field_exists()` giúp kiểm tra một trường có tồn tại trong một bảng nào đó không.

field_data()

Cú pháp

```
$this->db->field_data(string $tableName)
```

Phương thức `field_date()` trả về một mảng các đối tượng chứa thông tin về các trường trong một bảng nào đó.

```
$fields = $this->db->field_data('table_name');

foreach ($fields as $field)
{
   echo $field->name;
   echo $field->type;
   echo $field->max_length;
   echo $field->primary_key;
}
```

Trong đó:

- `name`: tên trường
- `max_length`: chiều dài dữ liệu tối đa của trường
- `primary_key`: có giá trị là 1 nếu trường này là khóa chính
- `type`: kiểu dữ liệu của trường

call_function()

Cú pháp

```
$this->db->call_function(string $functionName[, mixed $param1, mixed $param2…])
```

Phương thức `call_function()` cho phép ta sử dụng một hàm dựng sẵn của PHP nhưng không được hỗ trợ trong CodeIgniter. Ví dụ, để sửa dụng hàm `mysql_get_client_info()` vốn không được hỗ trợ trong CodeIgniter, ta thực hiện:

```
$this->db->call_function('get_client_info');
```

Khi sử dụng phương thức `call_function()`, ta không cần thiết phải truyền tiền tố `mysql_`. Các tiền tố này sẽ được tự động thêm vào, dựa vào hệ quản trị cơ sở dữ liệu đang kết nối. Điều này giúp nâng cao khả năng mở rộng của chương trình.

3.3.10. Quản trị cơ sở dữ liệu với Database Forge & Database Utility

Thư viện Database Forge & Database Utility được xây dựng nhằm giúp lập trình viên thực hiện các thao tác liên quan đến việc quản trị cơ sở dữ liệu, chẳng hạn như thêm/xóa/sửa cơ sở dữ liệu, thay đổi thông tin bảng…

Khai báo sử dụng Database Forge & Database Utility

Cũng giống như sử dụng các thư viện khác của CodeIgniter, để sử dụng thư viện Database Forge, ta sử dụng phương thức `load()` như sau:

```
$this->load->dbforge();
$this->load->dbutil();
```

Sau khi được khởi tạo, ta có thể sử dụng các phương thức của thư viện này bằng cách gọi:

```
$this->dbforge->some_function();
$this->dbutil->some_function();
```

Tạo cơ sở dữ liệu mới

Để tạo một cơ sở dữ liệu mới, ta sử dụng phương thức `create_database()` như sau:

```
$this->dbforge->create_database(string $databaseName)
```

Phương thức trả về `TRUE` nếu tạo cơ sở dữ liệu thành công, ngược lại trả về `FALSE`.

Xóa cơ sở dữ liệu

Để xóa một cơ sở dữ liệu, ta sử dụng phương thức `drop_database()` như sau:

```
$this->dbforge->drop_database(string $databaseName)
```

Phương thức trả về `TRUE` nếu xóa cơ sở dữ liệu thành công, ngược lại trả về `FALSE`.

Liệt kê các cơ sở dữ liệu

Để liệt kê các cơ sở dữ liệu hiện có trên máy chủ, ta sử dụng phương thức `list_databases()` như sau:

```
$this->dbforge->list_databases()
```

Phương thức trả về mảng chứa tên các cơ sở dữ liệu.

Tối ưu hóa bảng dữ liệu

Để thực hiện tối ưu hóa bảng dữ liệu, ta sử dụng phương thức `optimize_table()` như sau:

```
$this->dbutil->optimize_table(string $tableName)
```

Phương thức trả về `TRUE` nếu thực hiện thành công, ngược lại trả về `FALSE`. Lưu ý, chức năng này chỉ dành cho hệ quản trị cơ sở dữ liệu MySQL/MySQLi.

Sửa chữa bảng dữ liệu

Để thực hiện sửa chữa bảng dữ liệu, ta sử dụng phương thức `repair_table()` như sau:

```
$this->dbutil->optimize_table(string $tableName)
```

Phương thức trả về `TRUE` nếu thực hiện thành công, ngược lại trả về `FALSE`. Lưu ý, chức năng này chỉ dành cho hệ quản trị cơ sở dữ liệu MySQL/MySQLi.

Tối ưu hóa cơ sở dữ liệu

Để thực hiện tối ưu hóa tất cả cơ sở dữ liệu hiện có trên máy chủ, ta sử dụng phương thức `optimize_database()` như sau:

```
$this->dbutil->optimize_database()
```

Phương thức trả về mảng chứa các thông báo trạng thái cơ sở dữ liệu nếu thực hiện thành công, ngược lại trả về `FALSE`. Lưu ý, chức năng này chỉ dành cho hệ quản trị cơ sở dữ liệu MySQL/MySQLi.

Thêm bảng mới

Để thêm một bảng mới, ta sử dụng phương thức `create_table()` như sau:

```
$this->dbforge->create_table(string $tableName[, boolean $ifNotExists])
```

Biến tùy chọn `$ifNotExists` sẽ thêm vào lệnh `IF NOT EXISTS` khi tạo bảng. Phương thức trả về `TRUE` nếu thêm bảng thành công, ngược lại trả về `FALSE`.

Xóa bảng

Để xóa một bảng, ta sử dụng phương thức `drop_table()` như sau:

```
$this->dbforge->drop_table(string $tableName)
```

Đổi tên bảng

Để đổi tên bảng, ta sử dụng phương thức `rename_table()` như sau:

```
$this->dbforge->rename_table('old_table_name', 'new_table_name');
```

Thêm trường mới vào bảng

Để thêm trường mới vào bảng, ta sử dụng phương thức `add_field()` ngay sau khi gọi phương thức `create_table()`.

```
$this->dbforge->add_field(array $fields);
```

Các trường của một bảng được tạo thông qua một mảng quy ước, trong đó khóa của mảng là tên trường. Mỗi trường lại là một mảng có các khóa sau:

- `unsigned`: Nếu có giá trị TRUE, trường này có thuộc tính UNSIGNED
- `type`: Kiểu dữ liệu của trường, chẳng hạn như INT, VARCHAR, DATE…

- `constraint`: Một số kiểu dữ liệu, chẳng hạn như VARCHAR, yêu cầu phải cung cấp chiều dài dữ liệu.
- `default`: Giá trị mặc định của trường
- `null`: Nếu có giá trị TRUE, trường cho phép dữ liệu NULL, ngược lại là NOT NULL
- `auto_increment`: Nếu có giá trị TRUE, dữ liệu của trường sẽ được tự động tăng.

Ví dụ:

```
$fields = array(
                'blog_id' => array(
                                    'type' => 'INT',
                                    'constraint' => 5,
                                    'unsigned' => TRUE,
                                    'auto_increment' => TRUE
                                     ),
                'blog_title' => array(
                                    'type' => 'VARCHAR',
                                    'constraint' => '100',
                                     ),
                'blog_author' => array(
                                    'type' =>'VARCHAR',
                                    'constraint' => '100',
                                    'default' => 'King of Town',
                                     ),
                'blog_description' => array(
                                       'type' => 'TEXT',
                                       'null' => TRUE,
                                     ),
                );
```

Ngoài ra, ta cũng có thể thêm các trường bằng cách truyền trực tiếp câu lệnh SQL làm đối số cho hàm `add_field()` như sau:

```
$this->dbforge->add_field("label varchar(100) NOT NULL DEFAULT 'default
label'");
```

Thư viện Database Forge còn cho phép tạo trường ID một cách tự động bằng cách gọi:

```
$this->dbforge->add_field('id');
// Kết quả: id INT(9) NOT NULL AUTO_INCREMENT
```

Thêm khóa vào bảng

Để thêm khóa vào bảng, ta sử dụng phương thức `add_key()` như sau:

```
$this->dbforge->add_key(string $field, boolean $isPrimaryKey);
```

Phương thức `add_key()` phải được gọi ngay sau phương thức `create_table()`.

```
$this->dbforge->add_key('blog_name');
// Kết quả: KEY `blog_name` (`blog_name`)

$this->dbforge->add_key(array('blog_name', 'blog_label'));
// Kết quả: KEY `blog_name_blog_label` (`blog_name`, `blog_label`)
```

Biến tùy chọn `$isPrimaryKey` cho phép thêm khóa chính vào trường được chọn. Để chọn hai trường làm khóa chính, ta thực hiện gọi phương thức `add_key()` hai lần.

```
$this->dbforge->add_key('blog_id', TRUE);
// Kết quả: PRIMARY KEY `blog_id` (`blog_id`)

$this->dbforge->add_key('blog_id', TRUE);
$this->dbforge->add_key('site_id', TRUE);
// Kết quả: PRIMARY KEY `blog_id_site_id` (`blog_id`, `site_id`)
```

Thêm trường vào bảng đã tồn tại

Để thêm trường vào bảng đã tồn tại, ta sử dụng phương thức `add_column()` như sau:

```
$this->dbforge->add_column(string $tableName, array $fields);
```

Trong đó, mảng `$fields` là mảng kết hợp giống trong phương thức `add_field()`.

Xóa trường trong bảng đã tồn tại

Để xóa trường trong một bảng đã tồn tại, ta sử dụng phương thức `drop_column()` như sau:

```
$this->dbforge->drop_column(string $tableName, string $field);
```

Sửa thông tin trường

Để sửa thông tin trường trong một bảng đã tồn tại, ta sử dụng phương thức `modify_column()` như sau:

```
$this->dbforge->modify_column(string $tableName, array $modifiedField)
```

Cách sử dụng phương thức này giống như phương thức `add_field()`, ngoại trừ việc phương thức này sẽ thay đổi thông tin trường thay vì thêm mới. Mảng chứa thông tin của trường phải chứa khóa `name`. Ví dụ:

```
$fields = array(
                        'old_name' => array(
                                                'name' => 'new_name',
                                                'type' => 'TEXT',
                                            ),
);
$this->dbforge->modify_column('table_name', $fields);

// Kết quả: ALTER TABLE table_name CHANGE old_name new_name TEXT
```

Tạo CSV từ kết quả truy vấn

Để xuất kết quả truy vấn dưới dạng CSV, ta sử dụng phương thức `csv_from_result()` của thư viện Database Utility như sau:

```
$this->dbutil->csv_from_result($db_result, $delimiter, $newline);
```

Trong đó, `$db_result` là biến kết quả trả về sau khi thực hiện `SELECT`, `$delimiter` quy định ký tự được sử dụng để phân tách các trường giá trị, `$newline` quy định ký tự xuống dòng. Theo mặc định, CodeIgniter sử dụng tab để phân tách và ký tự `\n` để xuống dòng.

Tạo XML từ kết quả truy vấn

Để xuất kết quả truy vấn dưới dạng XML, ta sử dụng phương thức `xml_from_result()` của thư viện Database Utility như sau:

```
$this->dbutil->xml_from_result($db_result, array $config)
```

Trong đó `$db_result` là biến chứa kết quả trả về từ câu lệnh `SELECT`, `$config` là mảng bao gồm các thiết lập để xuất tập tin XML.

```
$this->load->dbutil();

$query = $this->db->query("SELECT * FROM mytable");

$config = array (
                  'root'    => 'root',
                  'element' => 'element',
                  'newline' => "\n",
                  'tab'     => "\t"
                );

echo $this->dbutil->xml_from_result($query, $config);
```

Theo đó, `root` là tên của tag gốc, `element` là tên của các tag thành phần, `newline` là ký tự được sử dụng khi xuống dòng, `tab` là ký tự được sử dụng để canh lề các tag.

Sao lưu cơ sở dữ liệu

Thư viện Database Utility cho phép sao lưu tất cả các bảng trong cơ sở dữ liệu hoặc từng bảng riêng biệt. Tập tin được sao lưu ở định dạng Zip hoặc Gzip. Lưu ý, chức năng này chỉ dành cho cơ sở dữ liệu MySQL. Do sự giới hạn về thời gian thực thi và bộ nhớ cấp phát của PHP, sao lưu cơ sở dữ liệu dung lượng lớn có thể không thực hiện được. Đối với những cơ sở dữ liệu như vậy, ta cần sử dụng công cụ sao lưu từ dòng lệnh hoặc yêu cầu người quản trị máy chủ thực hiện.

```
// Load thư viện Database Utility
$this->load->dbutil();

// Sao lưu toàn bộ cơ sở dữ liệu và gán vào biến
$backup =& $this->dbutil->backup();

// Load File helper để thực hiện ghi file
$this->load->helper('file');
write_file('/path/to/mybackup.gz', $backup);

// Sử dụng Download helper để gửi tập tin download về phía người dùng
$this->load->helper('download');
force_download('mybackup.gz', $backup);
```

Ta có thể thiết lập các tùy chọn cho việc sao lưu bằng cách truyền vào một mảng thiết lập.

```
$prefs = array(
                'tables'      => array('table1', 'table2'),
                'ignore'      => array(),
                'format'      => 'txt',
                'filename'    => 'mybackup.sql',
                'add_drop'    => TRUE,
                'add_insert'  => TRUE,
                'newline'     => "\n"
              );

$this->dbutil->backup($prefs);
```

Trong đó:

- `tables`: tên những bảng sẽ được sao lưu. Nếu mảng này rỗng, hệ thống sẽ sao lưu tất cả bảng.
- `ignore`: tên những bảng sẽ được bỏ qua.
- `format`: định dạng tập tin sẽ sao lưu. Database Utility hỗ trợ sao lưu gzip, zip hoặc txt. Mặc định là gzip.
- `filename`: tên tập tin sao lưu. Mặc định là ngày giờ hiện tại.
- `add_drop`: cho biết có thêm câu lệnh DROP TABLE vào tập tin sao lưu không.
- `add_insert`: cho biết có thêm câu lệnh INSERT vào tập tin sao lưu không.
- `newline`: ký tự sẽ được sử dụng khi xuống dòng. Hỗ trợ `\n`, `\r`, `\r\n`.

3.3.11. Bộ đệm cơ sở dữ liệu

Lớp Database Caching cho phép lưu trữ các truy vấn dưới dạng tập tin văn bản nhằm giảm tải cho máy chủ. Lớp này được khởi tạo một cách tự động khi chức năng caching được kích hoạt. Bằng cách sử dụng bộ đệm, ta có thể giảm thiểu số lần truy xuất trực tiếp vào cơ sở dữ liệu, nhờ đó nâng cao hiệu năng hệ thống. Tuy nhiên đối với các cơ sở

dữ liệu nhỏ với lưu lượng truy xuất dữ liệu thấp, sự cải thiện tốc độ sẽ khó nhận thấy hơn.

Tuy vậy, trong một số môi trường, ví dụ như các máy chủ chia sẻ (shared hosting), việc lưu bộ đệm có thể làm giảm tốc độ tải trang, vì ghi tập tin tốn khá nhiều tài nguyên. Do đó, để chức năng này hoạt động tốt, hệ thống cần được cài đặt trên một máy chủ riêng.

Kích hoạt bộ đệm

Để kích hoạt chức năng này, ta thực hiện theo ba bước sau:

1. Tạo một thư mục cho phép ghi trên máy chủ để lưu trữ các tập tin đệm.
2. Khai báo đường dẫn đến thư mục này trong tập tin `application/config/database.php`.
3. Kích hoạt chức năng lưu bộ đệm trên toàn hệ thống trong tập tin `application/config/database.php`.

Sau khi được kích hoạt, bộ đệm sẽ tự động vận hành khi tải một trang có truy vấn dữ liệu.

Cách thức hoạt động

Hệ thống lưu bộ đệm truy vấn của CodeIgniter tự vận hành khi trang được tải. Trong lần tải đầu tiên, đối tượng chứa các kết quả truy vấn sẽ được tuần tự hóa (serialize) và được lưu trữ thành tập tin văn bản trên máy chủ. Trong những lần truy cập tiếp theo, hệ thống sẽ lấy dữ liệu từ đối tượng trong tập tin văn bản này thay vì truy vấn trực tiếp vào cơ sở dữ liệu. Phương thức này giúp việc truy vấn cơ sở dữ liệu có thể được giảm xuống mức tối đa.

Chỉ duy nhất câu lệnh `SELECT` mới được lưu trong bộ đệm. Các câu lệnh `INSERT`, `UPDATE`, `DELETE`...sẽ không được lưu đệm vì chúng không trả về các kết quả truy vấn. Đồng thời, kết quả của các phương thức sau cũng không được lưu: `num_fields()`, `field_names()`, `field_data()`,`free_result()`.

Những tập tin đệm này sẽ được lưu trữ VÔ THỜI HẠN. Bất kỳ truy vấn nào cũng sẽ được lưu trong bộ đệm cho đến khi ta xóa chúng. CodeIgniter cho phép ta thực hiện dọn bộ đệm đối với từng trang cụ thể, hoặc dọn dẹp tất cả trong bộ đệm. CodeIgniter sẽ lưu trữ các tập tin đệm này vào các thư mục, với tên thư mục là tên controller và hàm tương ứng. Ví dụ: Kết quả truy vấn của trang `http://www.example.com/blog/comments` sẽ được lưu vào thư mục `blog+comments`.

Quản lý các tập tin đệm

Vì các tập tin đệm được lưu trữ vô thời hạn, ta cần xây dựng các quy trình dọn dẹp chúng. Chẳng hạn, khi người dùng thêm một sản phẩm mới vào hệ thống bán hàng, ta sẽ gọi bộ đệm tiến hành lưu tập tin mới và xóa các tập tin cũ. CodeIgniter hỗ trợ các phương thức sau để xử lý với bộ đệm.

cache_on() / cache_off()

Cú pháp

```
$this->db->cache_on();
$this->db->cache_off();
```

Cho phép bật/tắt chức năng lưu trữ bộ đệm, như trong ví dụ sau:

```
// Bật lưu trữ bộ đệm
$this->db->cache_on();
$query = $this->db->query("SELECT * FROM mytable");

// Tắt  bộ đệm, không lưu trữ truy vấn này
$this->db->cache_off();
$query = $this->db->query("SELECT * FROM members WHERE member_id =
'$current_user'");

// Bật lại lưu trữ bộ đệm
$this->db->cache_on();
$query = $this->db->query("SELECT * FROM another_table");
```

cache_delete()

Cú pháp

```
$this->db->cache_delete($controller, $method);
```

Phương thức `cache_delete()` cho phép xóa tập tin đệm tại một trang nào đó. Phương thức nhận tên của controller và method để xác định thư mục nào được xóa.

cache_delete_all()

Cú pháp

```
$this->db->cache_delete_all();
```

Phương thức `cache_delete_all()` sẽ xóa tất cả tập tin trong bộ đệm.

3.4. Email

Gửi email là một thao tác thường gặp khi xây dựng một ứng dụng web. Bản thân PHP có hỗ trợ hàm `mail()` để thực hiện gửi email. Nhưng để thực hiện những chức năng cao

cấp hơn, ta có thể sử dụng thư viện Email của CodeIgniter. Thư viện Email hỗ trợ những chức năng sau:

- Gửi email bằng nhiều giao thức: Mail, Sendmail và SMTP
- Có thể gửi cho nhiều người cùng lúc
- CC (carbon copy) và BCC (blind carbon copy)
- Gửi email dưới dạng HTML hoặc thuần văn bản
- Đính kèm tập tin
- Thiết lập độ ưu tiên
- Hỗ trợ wordwrap
- BCC Batch Mode, cho phép chia những danh sách email lớn thành những danh sách nhỏ hơn.
- Hỗ trợ công cụ tìm lỗi

Khai báo sử dụng thư viện Email

Cũng giống sử dụng các thư viện khác của CodeIgniter, để sử dụng thư viện Email, ta sử dụng phương thức `load()` như sau:

```
$this->load->library('email');
```

3.4.1. Thiết lập các tùy chọn

Để sử dụng thư viện Email, ta cần thiết lập các tùy chọn gửi email, chẳng hạn như giao thức, thông số của máy chủ SMTP…Các thiết lập này được lưu vào một mảng, và truyền vào phương thức `initialize()` để tiến hành khởi tạo. Chẳng hạn:

```
$config['protocol'] = 'sendmail';
$config['mailpath'] = '/usr/sbin/sendmail';
$config['charset'] = 'iso-8859-1';
$config['wordwrap'] = TRUE;

$this->email->initialize($config);
```

Ngoài ra, ta cũng có thể lưu các thiết lập vào mảng `$config` trong tập tin `application/config/email.php`. Khi đó các thiết lập này sẽ được gọi một cách tự động khi thư viện Email được khai báo sử dụng, ta không cần thiết phải gọi hàm `initialize()`.

Danh sách các tùy chọn:

Thiết lập	Giá trị mặc định	Tùy chọn	Mô tả
useragent	CodeIgniter	Không có	Cho biết chương trình

			được sử dụng để gửi mail
protocol	mail	mail, sendmail hoặc smtp	Giao thức được sử dụng để gửi mail
mailpath	/usr/bin/sendmail	Không có	Đường dẫn đến thư mục cài đặt Sendmail trên máy chủ
smtp_host	Không có	Không có	Địa chỉ của máy chủ SMTP
smtp_user	Không có	Không có	Tên tài khoản sử dụng máy chủ SMTP
smtp_pass	Không có	Không có	Mật khẩu của tài khoản
smtp_port	25	Không có	Cổng truy cập máy chủ SMTP
smtp_timeout	5	Không có	Thời gian trễ tạm ngưng khi truy cập máy chủ SMTP (tính bằng giây)
wordwrap	TRUE	TRUE/FALSE	Cho phép sử dụng wordwrap hay không
wrapchars	76		Số ký tự trên một dòng khi sử dụng wordwrap
mailtype	Text	text hoặc html	Định dạng của email
charset	utf8		Bảng mã được sử dụng
validate	FALSE	TRUE/FALSE	Có kiểm tra tính hợp lệ của địa chỉ email không
priority	3	1, 2, 3, 4, 5	Độ ưu tiên của email với 1 là cao nhất, 3 là bình thường và 5 là thấp nhất.
crlf	\n	\n, \r hoặc \r\n	Ký tự xuống dòng (sử dụng \r\n theo RFC 822)
newline	\n	\n, \r hoặc \r\n	Ký tự xuống dòng (sử dụng \r\n theo RFC 822)
bcc_batch_mode	FALSE	TRUE/FALSE	Có sử dụng BCC Batch

			Mode hay không
bcc_batch_size	200		Số địa chỉ email mỗi nhóm khi sử dụng BCC Batch Mode

3.4.2. Thực hiện gửi email

Sau khi khai báo sử dụng thư viện Email và thiết lập các tùy chọn cần thiết, ta có thể thực hiện gửi email như trong đoạn mã sau:

```
$this->load->library('email');

$this->email->from('your@example.com', 'Your Name');
$this->email->to('someone@example.com');
$this->email->cc('another@another-example.com');
$this->email->bcc('them@their-example.com');

$this->email->subject('Email Test');
$this->email->message('Testing the email class.');

$this->email->send();

echo $this->email->print_debugger();
```

3.4.3. Wordwrap

Chế độ wordwrap cho phép hiển thị văn bản trong một "khung" nhất định, giúp cho văn bản không tràn ra khỏi khung màn hình và thuận tiện cho người đọc. Theo RFC 822, chế độ wordwrap nên được kích hoạt. Thư viện Email cho phép thực hiện wordwrap trong nội dung email. Sau n ký tự (mặc định là 76, có thể được thiết lập bằng giá trị `wrapchars`), văn bản sẽ được đưa sang một hàng mới. Tuy nhiên, đối với những liên kết quá dài, khi thực hiện wordwrap sẽ làm cho chúng bị ngắt quãng, không thể click được. Để tránh điều này, thư viện Email sử dụng hai tag `{unwrap} {/unwrap}` để thông báo cho chương trình biết đoạn văn bản nào sẽ không được thực hiện wordwrap. Ví dụ

```
The text of your email that
gets wrapped normally.

{unwrap}http://example.com/a_long_link_that_should_not_be_wrapped.html{/unwrap}

More text that will be
wrapped normally.
```

3.4.4. Các phương thức

from()

Cú pháp

```
$this->email->from(string $email, string $name)
```

Phương thức `from()` cho phép thiết lập địa chỉ email và tên của người gửi.

reply_to()

Cú pháp

```
$this->email->reply_to(string $email, string $name)
```

Phương thức `reply_to()` cho phép thiết lập địa chỉ email sẽ nhận trả lời. Nếu phương thức này không được sử dụng, hệ thống sẽ tự nhận địa chỉ email trong phương thức from() làm địa chỉ nhận trả lời.

to()

Cú pháp

```
$this->email->to(string/array $emailAddresses)
```

Phương thức `to()` cho phép thiết lập những địa chỉ email sẽ được gửi. Các địa chỉ này được cách nhau bởi dấu phẩy (,) hoặc được lưu trong một mảng. Ví dụ:

```
$this->email->to('one@example.com, two@example.com, three@example.com');
$list = array('one@example.com', 'two@example.com', 'three@example.com');

$this->email->to($list);
```

cc()

Cú pháp

```
$this->email->cc(string/array $emailAddresses)
```

Phương thức `cc()` cho phép thiết lập những địa chỉ email sẽ được nhận bản sao khi gửi email (carbon copy). Người nhận sẽ thấy những địa chỉ được thiết lập bằng phương thức `cc()`. Cũng giống như phương thức `to()`, ta có thể truyền vào danh sách các địa chỉ email hoặc một mảng.

bcc()

Cú pháp

```
$this->email->bcc(string/array $emailAddresses)
```

Phương thức `bcc()` cho phép thiết lập những địa chỉ email sẽ được nhận bản sao khi gửi email, tuy nhiên người nhận sẽ không thấy những email được thiết lập trong phương thức `to()`, `cc()` và `bcc()` (blind carbon copy). Cũng giống như phương thức `to()`, ta có thể truyền vào danh sách các địa chỉ email hoặc một mảng.

subject()

Cú pháp

```
$this->email->subject(string $subject)
```

Phương thức `subject()` cho phép thiết lập tiêu đề của email.

message()

Cú pháp

```
$this->email->message(string $message)
```

Phương thức `message()` cho phép thiết lập nội dung của email.

set_alt_message()

Cú pháp

```
$this->email->set_alt_message(string $message)
```

Phương thức `set_alt_message()` cho phép thiết lập nội dung thay thế của email. Nếu email được gửi có dạng HTML nhưng người nhận không muốn hiển thị HTML vì lý do bảo mật, nội dung thay thế này sẽ được hiển thị. Nếu phương thức này không được sử dụng, CodeIgniter sẽ tự động tách bỏ các tag HTML trong phần nội dung email làm nội dung thay thế.

clear()

Cú pháp

```
$this->email->clear([boolean $clearAttachment])
```

Phương thức `clear()` sẽ xóa tất cả giá trị đã được thiết lập bởi các phương thức `from()`, `reply_to()`, `to()`, `cc()`, `bcc()`, `subject()`, `message()`. Phương thức này thường được sử dụng bên trong vòng lặp, giúp khởi động lại các giá trị sau mỗi lần lặp. Ví dụ:

```
foreach ($list as $name => $address)
{
    $this->email->clear();
```

```
    $this->email->to($address);
    $this->email->from('your@example.com');
    $this->email->subject('Here is your info '.$name);
    $this->email->message('Hi '.$name.' Here is the info you requested.');
    $this->email->send();
}
```

Nếu biến tùy chọn `$clearAttachment` có giá trị `TRUE`, tất cả tập tin đính kèm cũng bị xóa.

send()

Cú pháp

```
$this->email->send()
```

Phương thức `send()` sẽ thực hiện việc gửi email. Phương thức này trả về `TRUE` nếu gửi email thành công, ngược lại trả về `FALSE`.

attach()

Cú pháp

```
$this->email->attach(string $filePath)
```

Phương thức `attach()` sẽ đính kèm tập tin vào email. Phương thức này nhận đường dẫn tương đối đến tập tin trên máy chủ làm đối số thứ nhất. Nếu muốn đính kèm nhiều tập tin, ta gọi phương thức này nhiều lần.

print_debugger()

Cú pháp

```
$this->email->print_debugger()
```

Phương thức `print_debugger()` sẽ hiển thị những thông tin trả về từ phía server, email header cũng như nội dung email. Thường được sử dụng cho việc gỡ lỗi. Ví dụ:

```
if ( $this->email->send() == FALSE )
{
    $this->email->print_debugger();
}
```

3.5. Encryption

Trong một ứng dụng web, việc bảo mật thông tin của người sử dụng là điều bắt buộc. Đối với các ứng dụng thương mại điện tử, việc mã hóa thông tin khách hàng, chẳng hạn như số thẻ tín dụng, điện thoại, địa chỉ, email…quyết định sự sống còn của website.

Thư viện Encryption của CodeIgniter được xây dựng nhằm hỗ trợ lập trình viên thực hiện mã hóa/giải mã một cách đơn giản và hiệu quả.

Thư viện Encryption sử dụng cơ chế mã hóa đối xứng. Thông điệp cần mã hóa sẽ được tiền xử lý bằng cách thực hiện phép `XOR` với một đoạn hash ngẫu nhiên. Sau đó, kết quả thu được sẽ được mã hóa một lần nữa bởi thư viện Mcrypt. Nếu phiên bản PHP được cài đặt không kích hoạt thư viện Mcrypt, kết quả mã hóa vẫn cung cấp một mức độ bảo mật chấp nhận được cho các ứng dụng nhỏ. Trong trường hợp sử dụng thư viện Mcrypt, mức độ bảo mật được nâng cao rất nhiều.

Để sử dụng thư viện Encryption, ta khai báo phương thức `load()` như sau:

```
$this->load->library('encrypt');
```

Sau khi khai báo, ta có thể sử dụng các phương thức của thư viện Encryption bằng cách gọi:

```
$this->encrypt->some_function();
```

Lưu ý: Thông điệp sau khi mã hóa sẽ có chiều dài gấp khoảng 2.6 lần chiều dài của văn bản gốc. Do đó, cần lưu ý chọn kiểu dữ liệu lưu trữ phù hợp. Đối với hệ quản trị cơ sở dữ liệu MySQL, ta có thể dùng kiểu TEXT để lưu trữ thông điệp đã mã hóa.

3.5.1. Thiết lập khóa

Một ***khóa*** là một đoạn thông tin được sử dụng trong quá trình mã hóa/giải mã. Để giải mã một thông điệp, ta phải sử dụng khóa được dùng để mã hóa thông điệp đó. Do vậy, cần lựa chọn khóa một cách thận trọng. Nếu thay đổi khóa sẽ dẫn đến không thể giải mã những thông điệp đã tồn tại. Khóa nên có chiều dài 32 ký tự (128 bit), kết hợp ngẫu nhiên giữa số, chữ thường và chữ hoa.

Giá trị của khóa được thiết lập giá trị `$config['encryption_key']` trong tập tin `application/config/config.php`, hoặc ta cũng có thể đưa khóa vào khi gọi hàm mã hóa.

3.5.2. Các phương thức

encode()

Cú pháp

```
$this->encrypt->encode(string $message[, string $key])
```

Phương thức `encode()` sẽ mã hóa thông điệp truyền vào, sử dụng khóa mặc định được thiết lập trong tập tin `application/config/config.php`. Hoặc ta cũng có thể chọn một khóa khác và truyền vào làm đối số thứ hai. Ví dụ:

```
$msg = 'My secret message';
$key = 'super-secret-key';

$encrypted_string = $this->encrypt->encode($msg, $key);
```

decode()

Cú pháp

```
$this->encrypt->decode(string $message)
```

Phương thức `encode()` sẽ giải mã thông điệp truyền vào, sử dụng khóa mặc định được thiết lập trong tập tin `application/config/config.php`.

set_cipher()

Cú pháp

```
$this->encrypt->set_cipher(string $cipher)
```

Phương thức `set_cipher()` cho phép ta chọn thuật toán sẽ được sử dụng để mã hóa thông tin. Giá trị mặc định là `MCRYPT_RIJNDAEL_256` (thuật toán Rijndael 256 bits). Danh sách các thuật toán mã hóa có thể xem ở trang `http://vn2.php.net/manual/en/mcrypt.ciphers.php`.

set_mode()

Cú pháp

```
$this->encrypt->set_mode(string $mode)
```

Phương thức `set_mode()` cho phép thiết lập chế độ hoạt động của thư viện Mcrypt. Giá trị mặc định là `MCRYPT_MODE_ECB`. Danh sách các chế độ hoạt động khác có thể xem ở trang `http://vn2.php.net/manual/en/function.mcrypt-cfb.php`.

sha1()

Cú pháp

```
$this->encrypt->sha1(string $msg)
```

Phương thức `sha1()` cho phép mã hóa thông điệp theo thuật toán `SHA1`. Kết quả trả về là thông điệp mã hóa 160 bit. Cũng giống như `MD5`, `SHA1` là mã hóa bất đối xứng, thông điệp mã hóa không thể giải mã được.

3.6. Session

Thư viện Session của CodeIgniter giúp quản lý trạng thái của người dùng khi họ truy cập website. Các thông tin này được lưu trữ (và mã hóa) trong một tập tin cookie. Hoặc ta cũng có thể thiết lập lưu trữ session trong cơ sở dữ liệu để nâng cao tính bảo mật. Lưu ý, khái niệm session ở đây không liên quan đến session của PHP. CodeIgniter tạo ra dữ liệu session của riêng nó, nhằm cung cấp một cách thức quản lý linh động hơn.

Để sử dụng thư viện Session, ta khai báo như sau:

```
$this->load->library('session');
```

Sau khi khai báo, ta có thể sử dụng các phương thức của thư viện Session bằng cách gọi:

```
$this->session->some_function();
```

3.6.1. Thiết lập các tùy chọn

Các tùy chọn của thư viện Session được lưu trong tập tin `application/config/config.php`, bao gồm các giá trị sau:

Thiết lập	Giá trị mặc định	Tùy chọn	Mô tả
sess_cookie_name	ci_session	Không có	Tên của cookie sẽ chứa session
sess_expiration	7200		Thời hạn tồn tại của session này, được tính bằng giây. Mặc định là 2 giờ. Nếu có giá trị 0, session sẽ không bao giờ hết hạn.
sess_encrypt_cookie	FALSE	TRUE/FALSE	Cho biết có mã hóa dữ liệu trong session hay không
Sess_use_database	FALSE	TRUE/FALSE	Cho biết có lưu dữ liệu session vào cơ sở dữ liệu hay không
sess_table_name	ci_sessions		Tên của bảng sẽ được lưu

			session
sess_time_to_update	300		Thời gian (tính bằng giây) session sẽ được tái tạo
sess_match_ip	FALSE	TRUE/FALSE	Cho biết có so sánh địa chỉ IP khi đọc dữ liệu session hay không. Đối với IP động, nên cho tùy chọn này là FALSE.
sess_match_useragent	FALSE	TRUE/FALSE	Cho biết có so sánh thông tin trình duyệt khi đọc dữ liệu session hay không.

3.6.2. Cách thức hoạt động

Khi một trang được tải về, CodeIgniter sẽ kiểm tra có dữ liệu session hợp lệ trong cookie của người dùng hay không. Nếu session không tồn tại hoặc đã hết hạn, một session mới sẽ được tạo ra và lưu vào cookie của người dùng. Nếu session tồn tại, thông tin bên trong session sẽ được cập nhật và một session_id sẽ được tạo ra.

Thông tin session là một mảng chứa các giá trị sau:

- `session_id`: Một Session ID là một hash với độ tinh khiết cao, được mã hóa bằng thuật toán `MD5` mà được cập nhật sau mỗi 5 phút. Ta có thể thay đổi thời gian cập nhật này bằng cách thiết lập giá trị biến `$config['time_to_update']` trong tập tin `system/config/config.php`.
- `ip_address`: Địa chỉ IP của người dùng
- `user_agent`: User Agent: 50 ký tự đầu tiên về thông tin trình duyệt của người dùng
- `last_activity`: Timestamp chứa hoạt động cuối của người dùng

Các thông tin này sẽ được mã hóa nhằm nâng cao tính bảo mật, giúp ngăn chặn người khác thay đổi hay đọc session.

Lấy thông tin từ session

Để lấy thông tin từ session, ta sử dụng phương thức `userdata()` như sau:

```
$this->session->userdata(string $item);
```

Với `$item` là khóa của mảng chứa thông tin session. Chẳng hạn, để lấy địa chỉ IP của người dùng, ta sử dụng:

```
$this->session->userdata('ip_address');
```

Nếu khóa không tồn tại, phương thức trả về FALSE.

Thêm thông tin vào session

Để thêm một thông tin vào session hiện tại, ta sử dụng phương thức `set_userdata()` như sau:

```
$this->session->set_userdata(array $data);
```

Trong đó mảng `$data` có khóa là tên của thông tin và giá trị tương ứng của thông tin đó. Ví dụ:

```
$newdata = array(
                    'username'  => 'johndoe',
                    'email'     => 'johndoe@some-site.com',
                    'logged_in' => TRUE
                );

$this->session->set_userdata($newdata);
```

Hoặc ta cũng có thể thêm thông tin theo cách sau:

```
$this->session->set_userdata(string $item, string $value);
```

Xóa thông tin trong session

Để xóa thông tin trong session, ta sử dụng phương thức `unset_userdata()` như sau:

```
$this->session->unset_userdata(string $item);
```

Hoặc

```
$this->session->unset_userdata(array $items);
```

Trong trường hợp sử dụng mảng, ta khai báo mảng như sau:

```
$array_items = array('username' => '', 'email' => '');

$this->session->unset_userdata($array_items);
```

Xóa tất cả session

Để xóa tất cả session, ta sử dụng phương thức `sess_destroy()` như sau:

```
$this->session->sess_destroy();
```

3.6.3. Flashdata

CodeIgniter hỗ trợ "flashdata", là những session chỉ được sử dụng một lần và sẽ được xóa tự động. Flashdata được phân biệt với dữ liệu thường bởi tiền tố `flash_`. Để thêm một flashdata, ta sử dụng phương thức `set_flashdata()` như sau:

```
$this->session->set_flashdata(string $item, string $value);
```

Hoặc ta cũng có thể truyền vào một mảng như đối với phương thức `set_userdata()`.

Để lấy thông tin flashdata, ta sử dụng phương thức `flashdata()` như sau:

```
$this->session->flashdata('item');
```

Để giữ lại một flashdata, ta dùng phương thức keep_flashdata()

```
$this->session->keep_flashdata('item');
```

3.6.4. Lưu session vào cơ sở dữ liệu

Session được lưu trong cookie của người dùng chứa một Session ID. Ta không có cách nào tính xác thực của Session ID này. Bằng cách lưu session vào cơ sở dữ liệu, mỗi khi CodeIgniter phát hiện có thông tin về session trong cookie của người dùng, một truy vấn sẽ được thực hiện. Nếu Session ID của người dùng và trong cơ sở dữ liệu không trùng khớp, session sẽ bị hủy. Khi đó Session ID sẽ không được cập nhật, nó chỉ được tạo ra khi có session mới.

Để kích hoạt chức năng lưu session vào cơ sở dữ liệu, ta thiết lập biến `$config['sess_use_database']` có giá trị `TRUE` và biến `$config['sess_table_name']` trong tập tin `application/config/config.php`. Thông tin session được lưu trong một bảng có cấu trúc như sau:

```
CREATE TABLE IF NOT EXISTS  `ci_sessions` (
    session_id varchar(40) DEFAULT '0' NOT NULL,
    ip_address varchar(16) DEFAULT '0' NOT NULL,
    user_agent varchar(50) NOT NULL,
    last_activity int(10) unsigned DEFAULT 0 NOT NULL,
    user_data text NOT NULL,
    PRIMARY KEY (session_id)
);
```

4. Những helper hữu ích

4.1. Cookie

Cookie helper giúp ta thao tác với dữ liệu cookie trên máy người dùng, bao gồm các hàm sau:

set_cookie()

Cú pháp

```
set_cookie(array $data)
```

Cho phép thiết lập dữ liệu cookie trên máy người dùng. Mảng dữ liệu truyền vào có dạng như sau:

```
$cookie = array(
                    'name'   => 'The Cookie Name',
                    'value'  => 'The Value',
                    'expire' => '86500',
                    'domain' => '.some-domain.com',
                    'path'   => '/',
                    'prefix' => 'myprefix_',
                );

set_cookie($cookie);
```

Trong đó:

- `name`: tên của cookie
- `value`: giá trị của cookie này
- `expire`: thời gian tồn tại của cookie tính từ thời điểm hiện tại. Nếu thời gian này bằng 0, cookie sẽ được xóa.
- `domain`: domain của cookie
- `path`: đường dẫn của cookie, thường không cần thiết vì hàm đã thiết lập đường dẫn gốc.
- `prefix`: tiền tố, nhằm phân biệt với các cookie trùng tên khác.

Hoặc ta cũng có thể thiết lập cookie theo cú pháp sau:

```
set_cookie($name, $value, $expire, $domain, $path, $prefix);
```

get_cookie()

Cú pháp

```
get_cookie(string $cookieName[, boolean $xssFilter])
```

Lấy dữ liệu cookie từ máy người dùng. Hàm nhận tên cookie (bao gồm cả phần tiền tố) làm đối số đầu tiên. Nếu đối số thứ hai có giá trị `TRUE`, hàm sẽ thực hiện loại bỏ XSS giá trị đọc được. Hàm trả về `FALSE` nếu cookie không tồn tại.

delete_cookie()

Cú pháp

```
delete_cookie(string $cookieName)
```

Hàm `delete_cookie()` giúp xóa cookie trong máy người dùng. Hàm này tương đương với hàm `set_cookie()` với expiration bằng 0. Ta cũng có thể truyền gọi hàm này theo cú pháp:

```
delete_cookie($name, $domain, $path, $prefix)
```

4.2. File

File helper giúp ta thao tác với các tập tin, bao gồm các hàm sau:

read_file()

Cú pháp

```
read_file(string $path)
```

Hàm nhận đường dẫn đến tập tin làm đối số thứ nhất. Đường dẫn này có thể là đường dẫn tương đối hoặc tuyệt đối. Hàm trả về `FALSE` trong trường hợp thất bại.

write_file()

Cú pháp

```
write_file(string $path, string $data[, string $writeMode])
```

Hàm `write_file()` giúp ghi dữ liệu xuống tập tin. Nếu tập tin không tồn tại, CodeIgniter sẽ tự động tạo tập tin mới. Ta cũng có thể chọn chế độ ghi tập tin bằng cách truyền vào đối số thứ ba. Thông tin về các chế độ ghi tập tin có thể xem ở trang `http://vn2.php.net/manual/en/function.fopen.php`. Chế độ ghi mặc định là `wb`. Để hàm có thể hoạt động, yêu cầu tập tin phải được phép ghi (CHMOD 666, 777…) và thư mục chứa tập tin cũng phải được phép ghi.

delete_files()

Cú pháp

```
delete_files(string $path[, boolean $deleteDir])
```

Hàm `delete_files()` giúp xóa tất cả tập tin trong thư mục được truyền vào. Nếu đối số thứ hai có giá trị `TRUE`, tất cả thư mục bên trong đường dẫn được truyền vào cũng sẽ bị xóa.

get_filenames()

Cú pháp

```
get_filenames(string $path[, boolean $includePath])
```

Hàm `get_filenames()` trả về một mảng chứa tên của tất cả các tập tin trong thư mục được truyền vào. Nếu đối số thứ hai có giá trị `TRUE`, đường dẫn `$path` sẽ được thêm vào tên tập tin.

get_dir_file_info()

Cú pháp

```
get_dir_file_info(string $path)
```

Hàm `get_dir_file_info()` đọc thông tin thư mục được truyền vào, sau đó trả về mảng chứa tên tập tin, kích thước tập tin, ngày thay đổi và phân quyền. Thông tin về các thư mục con cũng được đọc.

get _file_info()

Cú pháp

```
get_file_info(string $path[, string $fileInformation])
```

Hàm `get_file_info()` đọc thông tin từ tập tin được truyền vào và trả về tên tập tin, kích thước, đường dẫn, ngày thay đổi. Đối số thứ hai cho phép lựa chọn thông tin nào sẽ được trả về, bao gồm: name (tên tập tin), server_path (đường dẫn đến tập tin), size (kích thước tập tin), date (ngày thay đổi), readable (tập tin có quyền đọc hay không), writable (tập tin có quyền ghi hay không), executable (tập tin có quyền thực thi hay không), fileperms (phân quyền của tập tin). Nếu tập tin không tìm thấy, hàm trả về `FALSE`.

symbolic_permissions()

Cú pháp

```
symbolic_permissions(string $perms)
```

Hàm `symbolic_permissions()` nhận phân quyền kiểu số (644, 777...kết quả trả về từ hàm `fileperms()`) và hiển thị dưới dạng ký tự.

Ví dụ:

```
echo symbolic_permissions(fileperms('./index.php'));

// -rw-r--r--
```

octal_permissions()

Cú pháp

```
octal_permissions(string $perms)
```

Hàm `octal_permissions()` nhận phân quyền kiểu kí tự và hiển thị dưới dạng số.

Ví dụ:

```
echo octal_permissions(fileperms('./index.php'));

// 644
```

5. Kết luận

Qua những tìm hiểu trên, ta thấy CodeIgniter là một nền tảng mạnh, cung cấp đầy đủ những thư viện và chức năng cần thiết để xây dựng một ứng dụng web với tốc độ nhanh mà vẫn đảm bảo hiệu năng hệ thống. So với các nền tảng khác, CodeIgniter không quá cồng kềnh, không làm lập trình viên giảm đi hứng thú khi sử dụng. Ngoài ra, cộng đồng người sử dụng CodeIgniter rất lớn, không ngừng đưa ra những plugin, helper hỗ trợ việc lập trình.

Dù cho CodeIgniter vẫn còn một số điểm hạn chế, nhưng về cơ bản, có thể nói CodeIgniter đang ngày càng nhận được sự ủng hộ từ phía các lập trình viên PHP. Phiên bản 2.0 của CodeIgniter đang trong giai đoạn phát triển beta, hứa hẹn sẽ đem đến những chức năng mới hữu ích và thú vị.

6. Danh mục từ viết tắt

- CRUD: Create Read Update Delete
- CSV: Comma Separated Values
- CRLF: Carriage Return [and] Line Feed
- EDP: Event-Driven Programming
- MVC: Model-View-Controller

- ORM: Object-Relational Mapping
- URI: Uniform Resource Identifier
- URL: Uniform Resource Location
- XML: Extensible Markup Language
- XSS: Cross Site Scripting

7. Tài liệu tham khảo

- CodeIgniter User Guide
  http://codeigniter.com/user_guide/
- Wikipedia – The Free Encyclopedia
  http://en.wikipedia.org/wiki/Model–view–controller
  http://en.wikipedia.org/wiki/Object-relational_mapping
  http://en.wikipedia.org/wiki/Event-driven_programming
  http://en.wikipedia.org/wiki/Active_record_pattern