Course - Laravel Framework

Cơ sở dữ liệu

Hầu hết mọi ứng dụng web hiện đại đều tương tác với cơ sở dữ liệu. Laravel làm cho việc tương tác với cơ sở dữ liệu trở nên cực kỳ đơn giản trên nhiều loại cơ sở dữ liệu được hỗ trợ bằng cách sử dụng SQL thô, trình tạo chuỗi truy vấn và Eloquent ORM.

Tags: database, co so du lieu, laravel

Giới thiệu

Hầu hết mọi ứng dụng web hiện đại đều tương tác với cơ sở dữ liệu. Laravel làm cho việc tương tác với cơ sở dữ liệu trở nên cực kỳ đơn giản trên nhiều loại cơ sở dữ liệu được hỗ trợ bằng cách sử dụng SQL thô, trình tạo chuỗi truy vấn và Eloquent ORM. Hiện tại, Laravel cung cấp hỗ trợ của bên thứ nhất cho năm cơ sở dữ liệu:

- MariaDB 10.2+ (Version Policy)
- MySQL 5.7+ (Version Policy)
- PostgreSQL 9.6+ (Version Policy)
- SQLite 3.8.8+
- SQL Server 2017+ (Version Policy)

Cấu hình

Cấu hình cho các dịch vụ cơ sở dữ liệu của Laravel nằm trong tập tin cấu hình *config/database.php* của ứng dụng của bạn. Trong tập tin này, bạn có thể xác định tất cả các kết nối cơ sở dữ liệu của mình, cũng như chỉ định kết nối nào nên được sử dụng theo mặc định. Hầu hết các tùy chọn cấu hình trong tập tin này được điều khiển bởi các giá trị của các biến môi trường ứng dụng của bạn. Các ví dụ cho hầu hết các hệ thống cơ sở dữ liệu được hỗ trợ của Laravel được cung cấp trong tập tin này.

Theo mặc định, cấu hình môi trường mẫu của Laravel đã sẵn sàng để sử dụng với Laravel Sail, là cấu hình Docker để phát triển các ứng dụng Laravel trên máy cục bộ của bạn. Tuy nhiên, bạn có thể tự do sửa đổi cấu hình cơ sở dữ liệu của mình phù hợp cho cơ sở dữ liệu cục bộ của bạn.

Cấu hình SQLite

Cơ sở dữ liệu SQLite được chứa trong một tập tin duy nhất trên hệ thống tập tin của bạn. Bạn có thể tạo cơ sở dữ liệu SQLite mới bằng cách sử dụng lệnh `touch` trong cửa sổ lệnh của mình: `touch database/database.sqlite`. Sau khi cơ sở dữ liệu đã được tạo, bạn có thể dễ dàng cấu hình các biến môi trường của mình để trỏ đến cơ sở dữ liệu này bằng cách đặt đường dẫn tuyệt đối đến cơ sở dữ liệu trong biến môi trường `DB_DATABASE`:

```
DB_CONNECTION=sqlite
DB_DATABASE=/absolute/path/to/database.sqlite
```

Để bật các ràng buộc khóa foreign cho các kết nối SQLite, bạn nên đặt biến môi trường `DB_FOREIGN_KEYS` thành `true`:

```
DB_FOREIGN_KEYS=true
```

Cấu hình Microsoft SQL Server

Để sử dụng cơ sở dữ liệu Microsoft SQL Server, bạn nên đảm bảo rằng bạn đã cài đặt các phần mở rộng PHP `sqlsrv` và `pdo_sqlsrv` cũng như bất kỳ thư viện nào mà chúng có thể yêu cầu, chẳng hạn như driver Microsoft SQL ODBC.

Cấu hình bằng URLs

Thông thường, các kết nối cơ sở dữ liệu được cấu hình bằng nhiều giá trị cấu hình như `host`, `database`, `username`, `password`, v.v. Mỗi giá trị cấu hình này có biến môi trường tương ứng riêng. Điều này có nghĩa là khi cấu hình thông tin kết nối cơ sở dữ liệu của bạn trên máy chủ đang xuất bản, bạn cần phải quản lý thêm một số biến môi trường.

Một số nhà cung cấp cơ sở dữ liệu được quản lý như AWS và Heroku cung cấp một "URL" cơ sở dữ liệu thống nhất chứa tất cả thông tin kết nối cho cơ sở dữ liệu trong một chuỗi duy nhất. URL cơ sở dữ liệu mẫu có thể trông giống như sau:

```
mysql://root:password@127.0.0.1/forge?charset=UTF-8
```

Các URL này thường tuân theo quy ước schema chuẩn:

```
driver://username:password@host:port/database?options
```

Để thuận tiện, Laravel hỗ trợ các URL này như một giải pháp thay thế cho việc cấu hình cơ sở dữ liệu của bạn với nhiều tùy chọn cấu hình. Nếu tùy chọn cấu hình `url` (hoặc biến môi trường `DATABASE_URL` tương ứng) có sẵn, nó sẽ được sử dụng để trích xuất kết nối cơ sở dữ liệu và thông tin xác thực.

Các kết nối đọc và viết

Đôi khi bạn có thể muốn sử dụng một kết nối cơ sở dữ liệu cho các câu lệnh `SELECT` và một kết nối khác cho các câu lệnh `INSERT`, `UPDATE` và `DELETE`. Laravel giúp việc này trở nên dễ dàng và các kết nối thích hợp sẽ luôn được sử dụng cho dù bạn đang sử dụng truy vấn thô, query builder hay Eloquent ORM.

Để xem cách cấu hình các kết nối đọc/ghi, hãy xem ví dụ sau:

```
'mysql' => [
  'read' => [
    'host' => [
      '192.168.1.1',
      '196.168.1.2',
    ],
  ],
  'write' => [
    'host' => [
      '196.168.1.3',
    ],
  ],
  'sticky' => true,
  'driver' => 'mysql',
  'database' => 'database',
  'username' => 'root',
  'password' => '',
  'charset' => 'utf8mb4',
  'collation' => 'utf8mb4_unicode_ci',
  'prefix' => '',
],
```

Lưu ý rằng có ba khóa đã được thêm vào mảng cấu hình: `read`, `write` và `sticky`. Các khóa `read` và `write` đều có các giá trị mảng chứa ít nhất một khóa: `host`. Phần còn lại của các tùy chọn cơ sở dữ liệu cho các kết nối `read` và `write` sẽ được hợp nhất từ mảng cấu hình `mysql` chính.

Bạn chỉ cần đặt các phần tử vào trong mảng `read` và `write` nếu bạn muốn ghi đè các giá trị từ mảng `mysql` chính. Vì vậy, trong trường hợp này, `192.168.1.1` sẽ được sử dụng làm máy chủ lưu trữ cho kết nối "read", trong khi `192.168.1.3` sẽ được sử dụng cho kết nối "write". Thông tin đăng nhập như `database`, `prefix`, `charset` và tất cả các tùy chọn khác trong mảng `mysql` chính sẽ được chia sẻ trên cả hai kết nối. Khi nhiều giá trị tồn tại trong mảng cấu hình host, một máy chủ cơ sở dữ liệu sẽ được chọn ngẫu nhiên cho mỗi yêu cầu.

Tùy chọn `sticky`

Tùy chọn `sticky` là một giá trị tùy chọn có thể được sử dụng để cho phép đọc ngay lập tức các record đã được ghi vào cơ sở dữ liệu trong suốt quá trình request hiện tại. Nếu tùy chọn `sticky` được bật và thao tác "write" đã được thực hiện đối với cơ sở dữ liệu trong suốt quá trình request hiện tại, thì bất kỳ thao tác "read" nào tiếp theo cũng sẽ sử dụng kết nối "write". Điều này đảm bảo rằng bất kỳ dữ liệu nào được ghi trong suốt quá trình request có thể được đọc lại ngay lập tức từ cơ sở dữ liệu trong cùng một request đó. Bạn cần xem xét kỹ trước khi quyết định xem đây có phải là hành vi mong muốn cho ứng dụng của mình hay không.

Chạy truy vấn SQL

Khi bạn đã cấu hình kết nối cơ sở dữ liệu của mình, bạn có thể chạy các truy vấn bằng cách sử dụng facade `DB`. facade `DB` cung cấp các phương thức cho từng loại truy vấn: `select`, `update`, `insert`, `delete` và `statement`.

Chạy truy vấn Select

Để chạy một truy vấn `SELECT` cơ bản, bạn có thể sử dụng phương thức `select` của facade `DB`:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
  /**
   * Show a list of all of the application's users.
   *
   * @return \Illuminate\Http\Response
   */
  public function index()
  {
    $users = DB::select('select * from users where active = ?', [1]);
```

```
    return view('user.index', ['users' => $users]);
  }
}
```

Đối số đầu tiên được truyền cho phương thức `select` là một chuỗi câu chứa cú pháp truy vấn SQL, trong khi đối số thứ hai là mảng các tham số mà sẽ được chèn tuần tự vào chuỗi truy vấn SQL trong đối số đầu tiên của phương thức `select`. Thông thường, đây là các giá trị của các câu điều kiện `where`. Liên kết tham số cũng bảo vệ chống lại lỗi SQL injection.

Phương thức `select` sẽ luôn trả về một mảng kết quả. Mỗi kết quả trong mảng sẽ là một đối tượng PHP `stdClass` đại diện cho một record từ cơ sở dữ liệu:

```
use Illuminate\Support\Facades\DB;

$users = DB::select('select * from users');

foreach ($users as $user) {
  echo $user->name;
}
```

Sử dụng các tham số có tên

Thay vì sử dụng dấu `?` để đại diện cho các tham số của bạn, thì bạn có thể thực hiện một truy vấn SQL bằng cách sử dụng các tham số được đặt tên rõ ràng:

```
$results = DB::select('select * from users where id = :id', ['id' => 1]);
```

Chạy lệnh insert

Để thực hiện một câu lệnh `insert`, thì bạn có thể sử dụng phương thức `insert` trên facade `DB`. Giống như `select`, phương thức này chấp nhận truy vấn SQL làm đối số đầu tiên và mảng các tham số là đối số thứ hai của nó:

```php
use Illuminate\Support\Facades\DB;

DB::insert('insert into users (id, name) values (?, ?)', [1, 'Marc']);
```

Chạy lệnh update

Phương pháp `update` nên được sử dụng để cập nhật các record hiện có trong cơ sở dữ liệu. Số hàng bị ảnh hưởng bởi câu lệnh được trả về theo phương thức:

```php
use Illuminate\Support\Facades\DB;

$affected = DB::update(
  'update users set votes = 100 where name = ?',
  ['Anita']
);
```

Chạy lệnh delete

Phương thức `delete` nên được sử dụng để xóa các record khỏi cơ sở dữ liệu. Giống như lệnh `update`, số record bị ảnh hưởng sẽ được trả về bởi phương thức:

```php
use Illuminate\Support\Facades\DB;

$deleted = DB::delete('delete from users');
```

Chạy một lệnh có tính chung chung

Có một vài câu lệnh cơ sở dữ liệu sẽ không trả về bất kỳ giá trị nào. Đối với trường hợp này, bạn có thể sử dụng phương thức `statement` trên facade `DB`:

```php
DB::statement('drop table users');
```

Chạy một lệnh chưa được chuẩn bị

Đôi khi bạn có thể muốn thực thi một câu lệnh SQL mà không ràng buộc bất kỳ giá trị nào. Bạn có thể sử dụng phương thức chưa chuẩn bị của facade `DB` để thực hiện điều này:

```
DB::unprepared('update users set votes = 100 where name = "Dries"');
```

> **Chú ý:** Vì các câu lệnh chưa chuẩn bị không ràng buộc các tham số, nên chúng có thể dễ bị SQL injection. Bạn không nên cho phép các giá trị do người dùng kiểm soát trong một câu lệnh chưa chuẩn bị.

Ngầm định chốt lệnh

Khi sử dụng các phương thức `statement` và `unprepared` của facade `DB` trong các giao dịch transaction, thì bạn phải cẩn thận để tránh các câu lệnh gây ra các vụ ngầm định chốt lệnh. Các câu lệnh này sẽ khiến cơ sở dữ liệu gián tiếp thực hiện toàn bộ giao dịch transaction, khiến Laravel không biết về mức giao dịch transaction của cơ sở dữ liệu. Một ví dụ về câu lệnh tạo một bảng cơ sở dữ liệu để chứng minh cho điều này như sau:

```
DB::unprepared('create table a (col varchar(1) null)');
```

Vui lòng tham khảo hướng dẫn sử dụng MySQL để biết danh sách tất cả các câu lệnh kích hoạt các vụ ngầm định chốt lệnh.

Sử dụng nhiều kết nối database

Nếu ứng dụng của bạn xác định nhiều kết nối trong tập tin cấu hình *config/database.php*, bạn có thể truy cập từng kết nối thông qua phương thức `connection` được cung cấp bởi facade `DB`. Tên kết nối được truyền vào cho phương thức connection phải tương ứng với một trong các kết nối được liệt kê trong tập tin cấu hình *config/database.php* của bạn hoặc được cấu hình trong thời gian chạy bằng hàm `config`:

```
use Illuminate\Support\Facades\DB;

$users = DB::connection('sqlite')->select(...);
```

Bạn có thể truy cập đối tượng PDO thô và cơ bản của một kết nối bằng cách sử dụng phương thức `getPdo` trên một đối tượng kết nối:

```php
$pdo = DB::connection()->getPdo();
```

Theo dõi các sự kiện truy vấn

Nếu bạn muốn chỉ định một hàm xử lý được gọi cho mỗi truy vấn SQL được thực thi bởi ứng dụng của bạn, bạn có thể sử dụng phương thức `listen` của facade `DB`. Phương thức này có thể hữu ích cho việc ghi log các truy vấn hoặc debug. Bạn có thể đăng ký hàm theo dõi truy vấn của mình trong phương thức `boot` của service provider:

```php
<?php

namespace App\Providers;

use Illuminate\Support\Facades\DB;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
  /**
   * Register any application services.
   *
   * @return void
   */
  public function register()
  {
    //
  }

  /**
   * Bootstrap any application services.
   *
   * @return void
   */
  public function boot()
  {
    DB::listen(function ($query) {
```

```
        // $query->sql;
        // $query->bindings;
        // $query->time;
      });
    }
}
```

Giao dịch transaction CSDL

Bạn có thể sử dụng phương thức giao dịch `transaction` được cung cấp bởi facade `DB` để chạy một tập hợp các hoạt động trong một giao dịch transaction cơ sở dữ liệu. Nếu một ngoại lệ được đưa ra trong hàm xử lý giao dịch transaction, giao dịch transaction sẽ tự động được khôi phục và ngoại lệ được đưa ra trở lại. Nếu hàm xử lý thực hiện thành công, giao dịch transaction sẽ tự động được chốt lệnh. Bạn không cần phải lo lắng về việc quay lại hoặc chốt lệnh theo cách thủ công trong khi sử dụng phương thức `transaction`:

```
use Illuminate\Support\Facades\DB;

DB::transaction(function () {
  DB::update('update users set votes = 1');

  DB::delete('delete from posts');
});
```

Xử lý các deadlocks

Phương thức `transaction` chấp nhận đối số thứ hai, đối số này là không bắt buộc, nó xác định số lần một giao dịch transaction nên được thử lại khi xảy ra bế tắc. Khi những nỗ lực này đã hết số lần thử lại, thì một ngoại lệ sẽ được đưa ra:

```
use Illuminate\Support\Facades\DB;

DB::transaction(function () {
  DB::update('update users set votes = 1');

  DB::delete('delete from posts');
```

```
}, 5);
```

Thao tác thủ công các giao dịch transaction

Nếu bạn muốn bắt đầu một giao dịch transaction theo cách thủ công và có toàn quyền kiểm soát đối với các lần khôi phục và chốt lệnh, thì bạn có thể sử dụng phương thức `beginTransaction` được cung cấp bởi facade `DB`:

```
use Illuminate\Support\Facades\DB;

DB::beginTransaction();
```

Bạn có thể khôi phục giao dịch transaction thông qua phương thức `rollBack`:

```
DB::rollBack();
```

Cuối cùng, bạn có thể thực hiện một giao dịch transaction thông qua phương thức `commit`:

```
DB::commit();
```

> Các phương thức giao dịch của facade `DB` sẽ kiểm soát các giao dịch cho cả query builder và Eloquent ORM.

Kết nối đến CLI cơ sở dữ liệu

Nếu bạn muốn kết nối với CLI của cơ sở dữ liệu của mình, bạn có thể sử dụng lệnh Artisan `db`:

```
php artisan db
```

Nếu cần, bạn có thể mô tả tên kết nối cơ sở dữ liệu để kết nối với kết nối nào không phải là kết nối mặc định:

```
php artisan db mysql
```