DATABASE: GETTING STARTED

1.1. Introduction

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 truy vấn (fluent query builder) và Eloquent ORM. Hiện tại, Laravel cung cấp hỗ trợ của bên thứ nhất cho bốn cơ sở dữ liệu:

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

1.1.1. Configuration

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

Theo mặc định, cấu hình môi trường (environment configuration) 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. Tuy nhiên, bạn có thể tự do sửa đổi cấu hình cơ sở dữ liệu nếu cần cho cơ sở dữ liệu cục bộ.

SQLite Configuration

Cơ sở dữ liệu SQLite được chứa trong một file duy nhất trên hệ thống filen. Có thể tạo cơ sở dữ liệu SQLite mới bằng cách sử dụng lệnh `touch` trong thiết bị đầu cuối của mình: `touch database/database.sqlite`. Sau khi cơ sở dữ liệu đã được tạo, có thể dễ dàng định cấu hình các biến môi trường 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 ngoại cho các kết nối SQLite, nên đặt biến môi trường `DB_FOREIGN_KEYS` thành `true`:

```
DB_FOREIGN_KEYS=true
```

Microsoft SQL Server Configuration

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

Configuration Using 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 định cấu hình thông tin kết nối cơ sở dữ liệu trên máy chủ, cần quản lý một số biến môi trường.

Một số nhà cung cấp dịch vụ quản trị cơ sở dữ liệu như AWS và Heroku cung cấp một "URL" cơ sở dữ liệu duy nhất chứa tất cả thông tin kết nối cho cơ sở dữ liệu trong một chuỗi duy nhất. Ví dụ một database URL 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 giản đồ chuẩn:

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

Trong đó:

- `Driver` là loại database muốn kết nối;
- `Username` là thông tin user đăng nhập vào database;
- `Password` là thông tin password đăng nhập vào database;
- `Host` là thông tin host của database;
- `Port` là port của datasbe;
- `Database` là tên database muốn kết nối;
- `Options` là các option muốn config kèm theo.

Để thuận tiện, Laravel hỗ trợ các URL này như một giải pháp thay thế cho việc định cấu hình cơ sở dữ liệu 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.

1.1.2. Read & Write Connections

Đôi khi 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 đang sử dụng truy vấn thô, `Query Builder` hay `Eloquent ORM`.

Để xem cách cấu hình các kết nối read/write, 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 ba keys đã được thêm vào mảng cấu hình: `read`, `write` và `sticky`. Các keys `read` và `write` có các giá trị mảng chứa một khóa duy nhất: `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.

Chỉ cần đặt các mục trong mảng `read` và `write` nếu 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 cơ sở dữ liệu, tiền tố, bộ ký tự 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`, máy chủ cơ sở dữ liệu sẽ được chọn ngẫu nhiên cho mỗi yêu cầu.

The sticky Option

Tùy chọn `sticky` là một giá trị tùy chọn (optional) có thể được sử dụng để cho phép đọc ngay lập tức các bản ghi đã được ghi vào cơ sở dữ liệu trong chu kỳ yêu cầu hiện tại. Nếu tùy chọn `sticky` được bật và thao

tác "rite" đã được thực hiện đối với cơ sở dữ liệu trong chu kỳ yêu cầu hiện tại, thì bất kỳ thao tác "read" nào tiếp theo sẽ sử dụng kết nối "write". Điều này đảm bảo rằng bất kỳ dữ liệu nào được ghi trong chu kỳ yêu cầu có thể được đọc lại ngay lập tức từ cơ sở dữ liệu trong cùng một yêu cầu đó. Hãy quyết định xem đây có phải là hành vi mong muốn cho ứng dụng hay không.

1.2. Running SQL Queries

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

Running A Select Query

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

```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 truy vấn SQL, trong khi đối số thứ hai là bất kỳ

ràng buộc tham số nào cần được liên kết với truy vấn. Thông thường, đây là các giá trị của các ràng buộc mệnh đề `where`. Parameter binding cung cấp bảo vệ chống lại SQL injection.

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

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

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

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

Using Named Bindings

Thay vì sử dụng `?` để đại diện cho các ràng buộc tham số, có thể thực hiện một truy vấn bằng cách sử dụng các ràng buộc có tên:

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

Running An Insert Statement

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

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

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

Running An Update Statement

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

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

$affected = DB::update(
    'update users set votes = 100 where name = ?',
```

```
    ['Anita']
);
```

Running A Delete Statement

Phương thức `delete` được sử dụng để xóa các bản ghi khỏi cơ sở dữ liệu. Giống như cập nhật, số dòng bị ảnh hưởng sẽ được trả về theo phương thức:

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

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

Running A General Statement

Một số câu lệnh cơ sở dữ liệu không trả về bất kỳ giá trị nào. Đối với các loại hoạt động này, có thể sử dụng phương thức `statement` trên `DB` facade:

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

Running An Unprepared Statement

Đôi khi muốn thực thi một câu lệnh SQL mà không ràng buộc bất kỳ giá trị nào. Có thể sử dụng phương thức `unprepared` của DB facade:

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

Vì các câu lệnh không chuẩn bị không ràng buộc các tham số, chúng có thể dễ SQL injection. Không bao giờ được phép cho phép các giá trị do người dùng kiểm soát trong một câu lệnh unprepared.

Implicit Commits

Khi sử dụngphương thức `statement` và `unprepared` của `DB` facade trong các giao dịch, phải cẩn thận để tránh các câu lệnh gây ra các cam kết ngầm (implicit commits). Các câu lệnh này sẽ khiến cơ sở dữ liệu thực hiện gián tiếp toàn bộ giao dịch, Laravel không biết về mức giao dịch của cơ sở dữ liệu. Một ví dụ về câu lệnh như vậy là tạo một bảng cơ sở dữ liệu:

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

Tham khảo hướng dẫn sử dụng MySQL để biết danh sách tất cả các câu lệnh a list of all statements.

1.2.1. Using Multiple Database Connections

Nếu ứng dụng định nghĩa nhiều kết nối trong file `config/database.php`, có thể truy cập từng kết nối thông qua phương thức `connection` được cung cấp bởi `DB` facade. Tên kết nối được chuyển cho phương thức `connection` phải tương ứng với một trong các kết nối được liệt kê trong file cấu hình `config/database.php` hoặc được định cấu hình trong thời gian chạy bằng `config` helper:

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

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

Có thể truy cập PDO instance của một kết nối bằng cách sử dụng phương thức `getPdo` trên một connection instance:

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

1.2.2. Listening For Query Events

Nếu muốn chỉ định một bao đóng được gọi cho mỗi truy vấn SQL được thực thi bởi ứng dụng, có thể sử dụng phương thức `listen` của `DB` facade. Phương thức này có thể hữu ích cho việc ghi nhật ký các truy vấn hoặc gỡ lỗi. Có thể đăng ký query listener closure trong phương thức `boot` của service provider:

```
<?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;
        });
    }
}
```

1.3. Database Transactions

Có thể sử dụng phương thức `transaction` được cung cấp bởi `DB` facade để chạy một tập hợp các hoạt động trong một giao dịch cơ sở dữ liệu. Nếu một ngoại lệ được đưa ra trong quá trình đóng giao dịch, giao dịch sẽ tự động được khôi phục lại. Nếu quá trình đóng thực hiện thành công, giao dịch sẽ tự động được cam kết. Không cần phải lo lắng về việc quay lại hoặc cam kết 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');
});
```

Handling Deadlocks

Phương thức `transaction` chấp nhận đối số thứ hai tùy chọn xác định số lần một giao dịch nên được thử lại khi xảy ra deadlock. Khi những nỗ lực này đã hết, 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);
```

Manually Using Transactions

If you would like to begin a transaction manually and have complete control over rollbacks and commits, you may use the `beginTransaction` method provided by the `DB` facade:

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

```
use Illuminate\Support\Facades\DB;
DB::beginTransaction();
```

Có thể khôi phục giao dịch thông qua phương thức `rollBack`:

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

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

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

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

1.4. Connecting To The Database CLI

Nếu muốn kết nối với database's CLI, có thể sử dụng lệnh Artisan `db`:

```
php artisan db
```

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

```
php artisan db mysql
```