DATABASE: SEEDING

1.1. Introduction

Seeder trong Laravel là class cho phép xử lý dữ liệu trong database. Class này sẽ hỗ trợ tạo ra data test, thay đổi cập nhật dữ liệu khi cần thiết.

Tất cả các seeder trong Laravel đều được đặt trong thư mục `database/seeders`. Mặc định thì Laravel đã định nghĩa sẵn cho chúng ta class `DatabaseSeeder` nằm sẵn trong thư mục trên. Từ lớp này, có thể sử dụng phương thức `call` để chạy các lớp seeder khác khi cần thiết.

1.2. Writing Seeders

Để tạo seeder, thực hiện lệnh `make:seeder` Artisan. Tất cả các seeder được tạo bởi framework sẽ được đặt trong thư mục `database/seeders`:

```
php artisan make:seeder UserSeeder
```

Theo mặc định, một lớp seeder chỉ chứa một default: `run`. Phương thức này được gọi khi lệnh `db:seed` Artisan được thực thi. Trong phương thức `run`, có thể chèn dữ liệu vào cơ sở dữ liệu của mình theo bất kỳ cách nào. Có thể sử dụng query builder để chèn dữ liệu theo cách thủ công hoặc có thể sử dụng Eloquent model factories.

Ví dụ, hãy modify lớp default `DatabaseSeeder` và thêm câu lệnh chèn cơ sở dữ liệu vào phương thức `run`:

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;
```

```
class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeders.
     *
     * @return void
     */
    public function run()
    {
        DB::table('users')->insert([
            'name' => Str::random(10),
            'email' => Str::random(10).'@gmail.com',
            'password' => Hash::make('password'),
        ]);
    }
}
```

1.2.1. Using Model Factories

Tất nhiên, việc chỉ định thủ công các thuộc tính cho từng seed model là rất phức tạp. Thay vào đó, có thể sử dụng các model factories để tạo ra một lượng lớn các bản ghi cơ sở dữ liệu một cách thuận tiện. Hãy xem lại tài liệu model factory documentation để tìm hiểu cách xác định các factories.

For example, let's create 50 users that each has one related post:

Ví dụ, hãy tạo 50 users mà mỗi user có một bài đăng liên quan:

```
use App\Models\User;

/**
 * Run the database seeders.
 *
 * @return void
 */
public function run()
```

```
{
    User::factory()
            ->count(50)
            ->hasPosts(1)
            ->create();
}
```

1.2.2. Calling Additional Seeders

Trong lớp `DatabaseSeeder`, có thể sử dụng phương thức `call` để thực thi các lớp seed bổ sung. Sử dụng phương thức `call` cho phép chia database seeding thành nhiều file để không có lớp single seeder nào trở nên quá lớn. Phương thức `call` chấp nhận một mảng các lớp seeder sẽ được thực thi:

```
/**
 * Run the database seeders.
 *
 * @return void
 */
public function run()
{
    $this->call([
        UserSeeder::class,
        PostSeeder::class,
        CommentSeeder::class,
    ]);
}
```

1.3. Running Seeders

Có thể thực hiện lệnh `db:seed` Artisan để tạo cơ sở dữ liệu. Theo mặc định, lệnh `db:seed` chạy lớp `Database\Seeders\DatabaseSeeder`, lớp này có thể lần lượt gọi các lớp seed khác. Tuy nhiên, có thể sử dụng tùy chọn `--class` để chỉ định một lớp seeder cụ thể để chạy riêng lẻ:

```
php artisan db:seed


php artisan db:seed --class=UserSeeder
```

Cũng có thể khởi tạo cơ sở dữ liệu của mình bằng cách sử dụng lệnh `migrate:fresh` kết hợp với tùy chọn `--seed`, tùy chọn này sẽ drop tất cả các bảng và chạy lại tất cả các migrations. Lệnh này rất hữu ích để xây dựng lại hoàn toàn cơ sở dữ liệu:

```
php artisan migrate:fresh --seed
```

Forcing Seeders To Run In Production

Một số thao tác seeding có thể khiến thay đổi hoặc mất dữ liệu. Để bảo vệ khỏi việc chạy các lệnh seeding dựa trên cơ sở dữ liệu, sẽ được nhắc xác nhận trước khi các seeders được thực thi trong môi trường `production`. Để buộc seeders chạy mà không có lời nhắc, hãy sử dụng cờ `--force`:

```
php artisan db:seed --force
```

REDIS

1.1. Introduction

Redis là một nền tảng tiên tiến cho phép lưu trữ các cặp dữ liệu kiểu key-value và tham khảo chúng rất nhanh, nó giống như một bộ đệm trong đó có thể làm việc với các phiên hoặc dữ liệu phải tồn tại trong một phiên mà không phải lưu trữ chúng trong cơ sở dữ liệu. Nó thường được gọi là máy chủ cấu trúc dữ liệu vì các khóa có thể chứa strings, hashes, lists, sets, và sorted sets.

Trước khi sử dụng Redis với Laravel, nên đặt và sử dụng phpredis PHP extension via PECL. Phần mở rộng phức tạp hơn để cài đặt so với các gói PHP "user-land" nhưng có thể mang lại hiệu suất tốt hơn cho các ứng dụng sử dụng nhiều Redis. Nếu đang sử dụng Laravel Sail, tiện ích mở rộng này đã được cài đặt trong vùng chứa Docker của ứng dụng.

Nếu không thể cài đặt phần phpredis extension, có thể cài đặt gói `predis/predis` thông qua Composer. Predis là một ứng dụng khách Redis được viết hoàn toàn bằng PHP và không yêu cầu bất kỳ phần mở rộng bổ sung nào:

```
composer require predis/predis
```

1.2. Configuration

Có thể định cấu hình cài đặt Redis của ứng dụng thông qua file cấu hình `config/database.php`. Trong file này, sẽ thấy một mảng redis chứa các máy chủ Redis được ứng dụng sử dụng:

```
'redis' => [

    'client' => env('REDIS_CLIENT', 'phpredis'),
```

```
    'default' => [
        'host' => env('REDIS_HOST', '127.0.0.1'),
        'password' => env('REDIS_PASSWORD', null),
        'port' => env('REDIS_PORT', 6379),
        'database' => env('REDIS_DB', 0),
    ],

    'cache' => [
        'host' => env('REDIS_HOST', '127.0.0.1'),
        'password' => env('REDIS_PASSWORD', null),
        'port' => env('REDIS_PORT', 6379),
        'database' => env('REDIS_CACHE_DB', 1),
    ],

],
```

Mỗi máy chủ Redis được xác định trong file cấu hình bắt buộc phải có name, host và port unless khi bạn xác định một URL duy nhất để đại diện cho kết nối Redis:

```
'redis' => [

    'client' => env('REDIS_CLIENT', 'phpredis'),

    'default' => [
        'url' => 'tcp://127.0.0.1:6379?database=0',
    ],

    'cache' => [
        'url' => 'tls://user:password@127.0.0.1:6380?database=1',
    ],

],
```

Configuring The Connection Scheme

By default, Redis clients will use the `tcp` scheme when connecting to your Redis servers; however, you may use TLS / SSL encryption by specifying a `scheme` configuration option in your Redis server's configuration array:

Theo mặc định, các máy khách Redis sẽ sử dụng `tcp` scheme khi kết nối với máy chủ Redis; tuy nhiên, có thể sử dụng mã hóa TLS/SSL bằng cách chỉ định tùy chọn cấu hình `scheme` trong mảng cấu hình máy chủ Redis:

```
'redis' => [

    'client' => env('REDIS_CLIENT', 'phpredis'),

    'default' => [
        'scheme' => 'tls',
        'host' => env('REDIS_HOST', '127.0.0.1'),
        'password' => env('REDIS_PASSWORD', null),
        'port' => env('REDIS_PORT', 6379),
        'database' => env('REDIS_DB', 0),
    ],

],
```

1.2.1. Clusters

Nếu ứng dụng đang sử dụng một cluster máy chủ Redis, nên xác định các cluster này trong một khóa `clusters` của cấu hình Redis. Khóa cấu hình này không tồn tại theo mặc định, vì vậy cần tạo nó trong file cấu hình `config/database.php` của ứng dụng:

```
'redis' => [

    'client' => env('REDIS_CLIENT', 'phpredis'),

    'clusters' => [
        'default' => [
```

```
            [
                'host' => env('REDIS_HOST', 'localhost'),
                'password' => env('REDIS_PASSWORD', null),
                'port' => env('REDIS_PORT', 6379),
                'database' => 0,
            ],
        ],
    ],

],
```

Theo mặc định, các clusters sẽ thực hiện sharding client-side trên các nút, cho phép gộp các nút và tạo ra một lượng lớn RAM có sẵn. Tuy nhiên, sharding client-side không xử lý chuyển đổi dự phòng; do đó, nó chủ yếu phù hợp với dữ liệu được lưu trong bộ nhớ cache tạm thời có sẵn từ một kho dữ liệu chính khác.

Nếu muốn sử dụng native Redis clustering thay vì client-side sharding, có thể chỉ định điều này bằng cách đặt giá trị cấu hình `options.cluster` thành `redis` trong file cấu hình `config/database.php` của ứng dụng:

```
'redis' => [

    'client' => env('REDIS_CLIENT', 'phpredis'),

    'options' => [
        'cluster' => env('REDIS_CLUSTER', 'redis'),
    ],

    'clusters' => [
        // ...
    ],

],
```

1.2.2. Predis

Nếu muốn ứng dụng của mình tương tác với Redis thông qua Predis package, nên chắc rằng giá trị của biến môi trường `REDIS_CLIENT` là `predis`:

```
'redis' => [

    'client' => env('REDIS_CLIENT', 'predis'),

    // Rest of Redis configuration...
],
```

Ngoài các tùy chọn cấu hình `host`, `port`, `database`, và `password` mặc định, Predis hỗ trợ các Connection parameters có thể được xác định cho từng máy chủ Redis. Để sử dụng các tùy chọn cấu hình bổ sung này, hãy thêm chúng vào cấu hình máy chủ Redis trong file cấu hình `config/database.php` của ứng dụng:

```
'default' => [
    'host' => env('REDIS_HOST', 'localhost'),
    'password' => env('REDIS_PASSWORD', null),
    'port' => env('REDIS_PORT', 6379),
    'database' => 0,
    'read_write_timeout' => 60,
],
```

The Redis Facade Alias

File cấu hình `config/app.php` của Laravel chứa một mảng `aliases` xác định tất cả các bí danh lớp sẽ được đăng ký bởi framework. Để thuận tiện, một mục bí danh được bao gồm cho mỗi facade do Laravel cung cấp; tuy nhiên, bí danh `Redis` bị vô hiệu hóa vì nó xung đột với tên lớp `Redis` do phpredis extension cung cấp. Nếu đang sử dụng Predis client và muốn bật bí danh này, có thể un-comment bí danh đó trong file cấu hình `config/app.php` của ứng dụng.

1.2.3. phpredis

Theo mặc định, Laravel sẽ sử dụng phpredis extension để giao tiếp với Redis. Ứng dụng khách mà Laravel sẽ sử dụng để giao tiếp với Redis được quy định bởi giá trị của tùy chọn cấu hình `redis.client`, thường phản ánh giá trị của biến môi trường `REDIS_CLIENT`:

```
'redis' => [
```

```
    'client' => env('REDIS_CLIENT', 'phpredis'),

    // Rest of Redis configuration...
],
```

Ngoài các tùy chọn cấu hình `host`, `port`, `database`, và `password` mặc định, phpredis hỗ trợ các tham số kết nối bổ sung sau: `name`, `persistent`, `prefix`, `read_timeout`, `retry_interval`, `timeout`, và `context`. Có thể thêm bất kỳ tùy chọn nào trong số này vào cấu hình máy chủ Redis của mình trong file cấu hình `config/database.php`:

```
'default' => [
    'host' => env('REDIS_HOST', 'localhost'),
    'password' => env('REDIS_PASSWORD', null),
    'port' => env('REDIS_PORT', 6379),
    'database' => 0,
    'read_timeout' => 60,
    'context' => [
        // 'auth' => ['username', 'secret'],
        // 'stream' => ['verify_peer' => false],
    ],
],
```

1.3. Interacting With Redis

Có thể tương tác với Redis bằng cách gọi nhiều phương thức khác nhau trên `Redis` facade. `Redis` facade hỗ trợ các phương thức động, có nghĩa là có thể gọi bất kỳ Redis command nào trên facade và lệnh sẽ được truyền trực tiếp đến Redis.

Ví dụ, gọi lệnh Redis `GET` bằng cách gọi phương thức `get` trên `Redis` facade:

```
<?php

namespace App\Http\Controllers;
```

```php
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Redis;

class UserController extends Controller
{
    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        return view('user.profile', [
            'user' => Redis::get('user:profile:'.$id)
        ]);
    }
}
```

Có thể gọi bất kỳ lệnh nào của Redis trên `Redis` facade. Laravel sử dụng các phương thức magic để truyền các lệnh đến Redis server. Nếu một lệnh Redis yêu cầu các đối số, nên truyềnn các đối số đó đến phương thức tương ứng của facade:

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

Redis::set('name', 'Taylor');

$values = Redis::lrange('names', 5, 10);
```

Ngoài ra, có thể truyền các lệnh tới máy chủ bằng cách sử dụng phương thức `Redis` facade's `command`, phương thức này chấp nhận tên của lệnh làm đối số đầu tiên và một mảng giá trị làm đối số thứ hai của nó:

```php
$values = Redis::command('lrange', ['name', 5, 10]);
```

Using Multiple Redis Connections

File cấu hình `config/database.php` của ứng dụng cho phép xác định nhiều Redis connections / servers. Có thể nhận được kết nối đến một kết nối Redis cụ thể bằng cách sử dụng phương thức `connection` của `Redis` facade:

```
$redis = Redis::connection('connection-name');
```

Để có được một instance của kết nối Redis mặc định, có thể gọi phương thức `connection` mà không có bất kỳ đối số bổ sung nào:

```
$redis = Redis::connection();
```

1.3.1. Transactions

Phương thức `transaction` của `Redis` facade cung cấp một wrapper xung quanh các Redis' native `MULTI` and `EXEC` commands. Phương thức `transaction` chấp nhận một closure làm đối số duy nhất của nó. Closure này sẽ nhận được một instance Redis connection và có thể đưa ra bất kỳ lệnh nào nó muốn cho instance này. Tất cả các lệnh Redis được đưa ra trong closure sẽ được thực hiện trong một single, atomic transaction:

```
use Illuminate\Support\Facades\Redis;

Redis::transaction(function ($redis) {
    $redis->incr('user_visits', 1);
    $redis->incr('total_visits', 1);
});
```

Lua Scripts

Phương thức `eval` cung cấp một phương pháp khác để thực hiện nhiều lệnh Redis trong một hoạt động single, atomic. Tuy nhiên, phương thức `eval` có lợi ích là có thể tương tác và kiểm tra các giá trị chính của Redis trong quá trình hoạt động đó. Các tập lệnh Redis được viết Lua programming language.

Lúc đầu, phương thức `eval` có thể hơi khó hiểu, hãy xem một ví dụ sau. Phương thức `eval` mong đợi một số đối số. Đầu tiên, truyền Lua script (dưới dạng một string) vào phương thức. Thứ hai, truyền số khóa (dưới dạng số nguyên) mà tập lệnh tương tác với. Thứ ba, truyền tên của các khóa đó. Cuối cùng, có thể truyền bất kỳ đối số bổ sung nào khác cần truy cập trong script.

Ví dụ, sẽ tăng bộ đếm, kiểm tra giá trị mới của nó và tăng bộ đếm thứ hai nếu giá trị của bộ đếm đầu tiên lớn hơn năm. Cuối cùng, sẽ trả về giá trị của bộ đếm đầu tiên:

```
$value = Redis::eval(<<<'LUA'
    local counter = redis.call("incr", KEYS[1])

    if counter > 5 then
        redis.call("incr", KEYS[2])
    end

    return counter
LUA, 2, 'first-counter', 'second-counter');
```

Tham khảo Redis documentation để biết thêm thông tin về Redis scripting.

1.3.2. Pipelining Commands

Đôi khi cần thực hiện hàng chục lệnh Redis. Sử dụng phương thức `pipeline`. Phương thức `pipeline` chấp nhận một đối số: một closure nhận một instance Redis. Có thể đưa ra tất cả các lệnh của mình cho instance Redis này và tất cả chúng sẽ được gửi đến máy chủ Redis cùng một lúc. Các lệnh sẽ vẫn được thực hiện theo thứ tự:

```
use Illuminate\Support\Facades\Redis;

Redis::pipeline(function ($pipe) {
    for ($i = 0; $i < 1000; $i++) {
        $pipe->set("key:$i", $i);
    }
});
```

1.4. Pub / Sub

Laravel cung cấp một giao diện thuận tiện cho các lệnh `publish` và `subscribe` của Redis. Các lệnh Redis này cho phép nghe tin nhắn trên một "channel" nhất định. Có thể xuất bản tin nhắn lên kênh từ một ứng dụng khác hoặc thậm chí sử dụng ngôn ngữ lập trình khác, cho phép giao tiếp dễ dàng giữa các applications và processes.

Trước tiên, hãy thiết lập channel listener bằng phương thức `subscribe`. Đặt lệnh gọi phương thức này trong một lệnh Artisan command vì việc gọi phương thức `subscribe` bắt đầu một quá trình chạy lâu dài:

```php
<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;
use Illuminate\Support\Facades\Redis;

class RedisSubscribe extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'redis:subscribe';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Subscribe to a Redis channel';

    /**
     * Execute the console command.
     *
     * @return mixed
     */
    public function handle()
    {
        Redis::subscribe(['test-channel'], function ($message) {
            echo $message;
        });
```

```
    }
}
```

Bây giờ, có thể xuất bản tin nhắn lên kênh bằng phương thức `publish`:

```
use Illuminate\Support\Facades\Redis;

Route::get('/publish', function () {
    // ...

    Redis::publish('test-channel', json_encode([
        'name' => 'Adam Wathan'
    ]));
});
```

Wildcard Subscriptions

Sử dụng phương thức `psubscribe`, có thể đăng ký một wildcard channel, điều này có thể hữu ích để xem tất cả các thông báo trên tất cả các kênh. Tên kênh sẽ được chuyển làm đối số thứ hai cho closure được cung cấp:

```
Redis::psubscribe(['*'], function ($message, $channel) {
    echo $message;
});

Redis::psubscribe(['users.*'], function ($message, $channel) {
    echo $message;
});
```