Database: Seeding

# Introduction

Laravel includes a simple method of seeding your database with test data using seed classes. All seed classes are stored in the `database/seeds` directory. Seed classes may have any name you wish, but probably should follow some sensible convention, such as `UsersTableSeeder`, etc. By default, a `DatabaseSeeder` class is defined for you. From this class, you may use the `call` method to run other seed classes, allowing you to control the seeding order.

Writing Seeders

To generate a seeder, execute the `make:seeder` Artisan command. All seeders generated by the framework will be placed in the `database/seeds` directory:

```
php artisan make:seeder UsersTableSeeder
```

A seeder class only contains one method by default: `run`. This method is called when the `db:seed` Artisan command is executed. Within the `run` method, you may insert data into your database however you wish. You may use the query builder to manually insert data or you may use Eloquent model factories.

> Mass assignment protection is automatically disabled during database seeding.

As an example, let's modify the default `DatabaseSeeder` class and add a database insert statement to the `run` method:

```
<?php

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

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

> You may type-hint any dependencies you need within the `run` method's signature. They will automatically be resolved via the Laravel service container.

Using Model Factories

Of course, manually specifying the attributes for each model seed is cumbersome. Instead, you can use model factories to conveniently generate large amounts of database records. First, review the model factory documentation to learn how to define your factories. Once you have defined your factories, you may use the `factory` helper function to insert records into your database.

For example, let's create 50 users and attach a relationship to each user:

```
/**
 * Run the database seeds.
 *
 * @return void
 */
public function run()
{
    factory(App\User::class, 50)->create()->each(function ($user) {
        $user->posts()->save(factory(App\Post::class)->make());
    });
}
```

Calling Additional Seeders

Within the `DatabaseSeeder` class, you may use the `call` method to execute additional seed classes. Using the `call` method allows you to break up your database seeding into multiple files so that no single seeder class becomes overwhelmingly large. Pass the name of the seeder class you wish to run:

```
/**
 * Run the database seeds.
 *
 * @return void
 */
public function run()
{
    $this->call([
        UsersTableSeeder::class,
        PostsTableSeeder::class,
        CommentsTableSeeder::class,
    ]);
}
```

Running Seeders

Once you have written your seeder, you may need to regenerate Composer's autoloader using the `dump-autoload` command:

```
composer dump-autoload
```

Now you may use the `db:seed` Artisan command to seed your database. By default, the `db:seed` command runs the `DatabaseSeeder` class, which may be used to call other seed classes. However, you may use the `--class` option to specify a specific seeder class to run individually:

```
php artisan db:seed

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

You may also seed your database using the `migrate:fresh` command, which will drop all tables and re-run all of your migrations. This command is useful for completely re-building your database:

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

Forcing Seeders To Run In Production

Some seeding operations may cause you to alter or lose data. In order to protect you from running seeding commands against your production database, you will be prompted for confirmation before the seeders are executed. To force the seeders to run without a prompt, use the `--force` flag:

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

Become a Laravel Partner

Laravel Partners are elite shops providing top-notch Laravel development and consulting. Each of our partners can help you craft a beautiful, well-architected project.

Our Partners

Laravel

Highlights

- Release Notes
- Getting Started
- Routing
- Blade Templates
- Authentication
- Authorization
- Artisan Console
- Database
- Eloquent ORM
- Testing

Resources

- Laracasts
- Laravel News
- Laracon
- Laracon EU
- Laracon AU
- Jobs
- Certification
- Forums

Partners

- Vehikl
- Tighten Co.
- Kirschbaum
- Byte 5
- 64Robots
- Cubet
- DevSquad
- Ideil
- Cyber-Duck
- ABOUT YOU
- Become A Partner

Ecosystem

- Vapor
- Forge
- Envoyer
- Horizon
- Lumen
- Nova
- Echo
- Valet
- Mix
- Spark
- Cashier
- Homestead
- Dusk
- Passport

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in most web projects.