Les opérations CRUD avec Eloquent Laravel 1-3

1. Objectifs

- Qu'est-ce que le "seeder" de base de données dans Laravel 8?
- Comment créer un "seeder" dans Laravel 8?

2. Présentation

- L'une des caractéristiques les plus importantes du framework laravel est qu'il est livré avec **ORM** (**O**bject **R**elation **M**apping) intégré appelé Eloquent **ORM**.
- Eloquent Object Relation Mapper est une fonctionnalité importante du framework Laravel, considérée comme une fonctionnalité puissante et coûteuse de Laravel.
- L'**ORM** fonctionne avec des objets de base de données et est utilisé pour établir des relations avec des tables de base de données. Chaque table de la base de données est mappée avec un **modèle** éloquent particulier.
- L'objet modèle contient diverses méthodes pour récupérer et mettre à jour les données de la table de base de données.
- Eloquent ORM peut être utilisé avec plusieurs bases de données en implémentant `ActiveMethod`. Cette fonctionnalité simplifie les tâches liées à la base de données, telles que la définition de relations, en définissant les tables de la base de données.
- Les opérations **CRUD** sous le mappeur objet-relationnel Eloquent (**ORM**) permettent aux développeurs Laravel de travailler plus facilement avec plusieurs bases de données.
- Eloquent **ORM** rend facile l'exécution des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) sur le modèle Laravel.
- Il effectue des opérations de création, de récupération, de mise à jour et de suppression (CRUD), ainsi que les mappages de modèles d'objets sur des tables de base de données. Il gère toutes les interactions de base de données requises pour les opérations CRUD.

3. Modèles éloquents

- Eloquent est un framework ORM qui est utilisé dans Laravel pour faciliter et accélérer le développement Web en se concentrant sur une approche orientée objet plutôt que sur l'écriture de requêtes SQL simples.
- Chaque table de base de données peut avoir le modèle Eloquent correspondant. Eloquent Model représente des entités de base de données et peut être utilisé pour

apcpedagogie.com

interroger des données ainsi que pour *insérer* et *mettre à jour* des données dans les tables.

- **Syntaxe:**
  - **`php artisan make:model NomTable`**
  - Cette commande créera le modèle de "NomTable" dans le répertoire de l'application comme indiqué ci-dessous.

4. ## Activité (Créer un modèle et une migration)

   - Créer un nouveau modèle "Voiture" en utilisant la commande artisan suivante: `php artisan make:model Voiture -m`
   - Ouvrir le fichier de modèle nouvellement généré, à ce stade, votre modèle devrait ressembler à ceci:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Voiture extends Model
{
    use HasFactory;
}
```

5. Commencer par définir les champs de votre base de données pour notre table voitures dans notre fichier de migration comme suit:

```
public function up()
    {
        Schema::create('voitures', function (Blueprint $table) {
            $table->string('plaque');
            $table->string('marque');
            $table->string('modele');
            $table->string('couleur');
            $table->string('cylindree');
            $table->string('transmission');
            $table->float('prix');
            $table->timestamps();
```

apcpedagogie.com

```
            $table->primary('plaque'); /* crée une clé primaire */
        });
    }
```

> *Chaque modèle étend le modèle **Model** par défaut de Laravel, ce qui lui confère déjà des éléments de configuration et de fonctionnement. Par exemple, le nom de la table associée au modèle est automatique le nom du modèle mis au pluriel. Dans notre cas, étant donné que nous avons nommé notre modèle `Voiture`, Laravel va automatiquement lier notre modèle à la table `voitures`.*

- Ajoutez la colonne "type" de type string, n'utilisez pas la commande **rollback**.
- **`php artisan migrate:refresh --path=database/migrations/2020_12_23_094127_clients.php`**

La population (seeding)

- Laravel inclut la possibilité d'amorcer votre base de données avec des données de test à l'aide de classes d'amorçage. Toutes les classes de départ sont stockées dans le répertoire **database/seeders**.
- Par défaut, une classe **DatabaseSeeder** est définie pour vous. À partir de cette classe, vous pouvez utiliser la méthode **`call`** pour exécuter d'autres classes d'amorçage, ce qui vous permet de contrôler l'ordre d'amorçage.

- **Écriture des seeders**
    - Pour générer un seeder, exécutez la commande Artisan **`make:seeder`**: `php artisan make:seeder UsersTableSeeder`.
    - Tous les seeders générés par le Framework seront placés dans le répertoire **database/seeders**:

- **Méthodes utilisées**
    - Une classe seedere ne contient qu'une seule méthode par défaut: `run`. Cette méthode est appelée lorsque la commande Artisan `db:seed` est exécutée. Dans la méthode `run`, vous pouvez insérer des données dans votre base de données comme vous le souhaitez.

- **Génération**
    - Vous pouvez utilisez la commande suivante pour insérer des données dans les tables de votre base de données

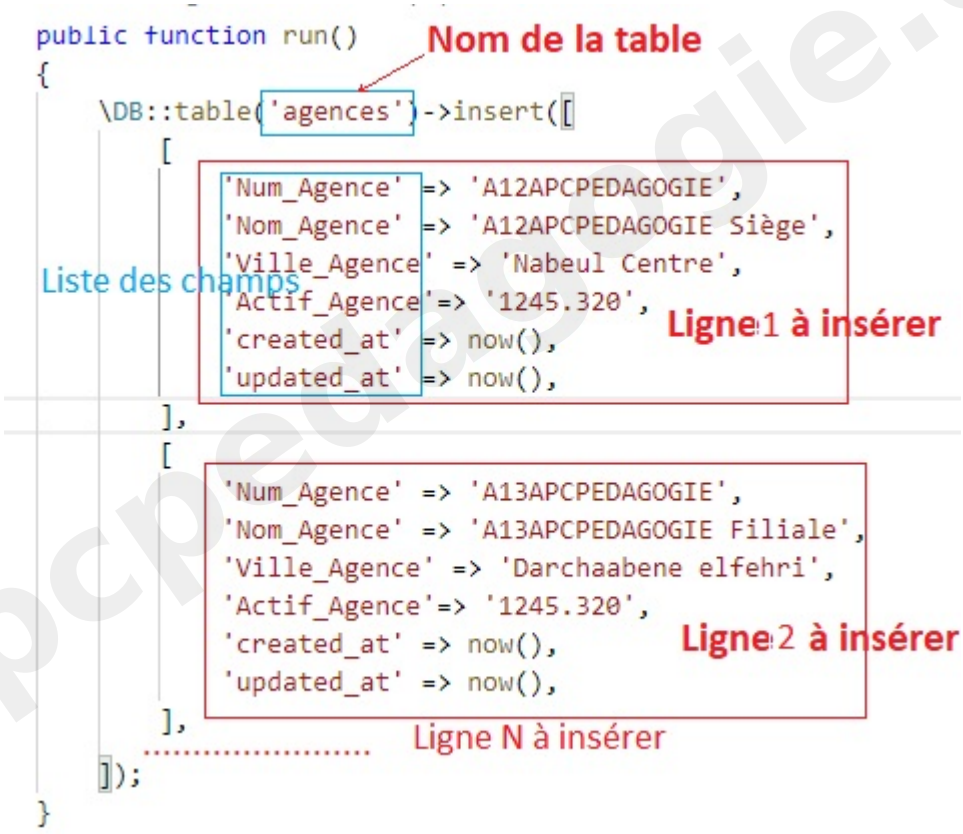
apcpedagogie.com

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

- **Exemple**

  - À titre d'exemple, modifions la classe `DatabaseSeeder` par défaut et ajoutons une instruction d'insertion de base de données à la méthode `run`:

```
public function run()
    {
        DB::table('users')->insert([
            'name' => str_random(10),
            'email' => str_random(10).'@gmail.com',
            'password' => bcrypt('secret'),
        ]);
    }
```



- **Application**

1. Créer, en utilisant la ligne de commande, un Seeder nommé **VoituresTableSeeder**.
2. Ajouter, dans le Seeder, le code nécessaire permettant de remplir la table **voitures** par des données de test.

3. Exécuter le Seeder puis vérifier la table dans l'interface phpmyadmin
4. **`php artisan db:seed --class=VoituresTableSeeder`**

- ## Application

  - Soit les deux fichiers de migration suivant:

1.
```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration {

	public function up()
	{
		Schema::create('users', function(Blueprint $table)
		{
			$table->increments('id');
			$table->string('name')->unique();
			$table->string('email')->unique();
			$table->string('password', 60);
			$table->boolean('admin')->default(false);
			$table->rememberToken();
			$table->timestamps();
		});
	}

	public function down()
	{
		Schema::drop('users');
	}

}
```

2.
```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
```

apcpedagogie.com

```
class CreatePostsTable extends Migration {

    public function up()
    {
    Schema::create('posts', function(Blueprint $table) {
$table->increments('id');
$table->timestamps();
$table->string('titre', 80);
$table->text('contenu');
$table->integer('user_id')->unsigned();
$table->foreign('user_id')
    ->references('id')
    ->on('users')
    ->onDelete('restrict')
    ->onUpdate('restrict');
});
}

public function down()
{
Schema::table('posts', function(Blueprint $table) {
$table->dropForeign('posts_user_id_foreign');
});
Schema::drop('posts');
}

}
```

- Utilisez PHPMyAdmin pour créer une Base de données dont le nom "UsersPosts" (Messages des utilisateurs)
- Utilisez la commande artisanal **`php artisan make:migration`** pour créer les fichiers de migration ci-dessus (changer le fichier s'il existe)

- Configurez la base de données dans votre projet (**MessagesUtilisateurs**) Laravel
- Modifiez le fichier **config/database.php**, dans l'array '**mysql**', deux lignes a modifier:
- Lancez la migration : **`php artisan migrate`**
- Vous devez ainsi vous retrouver avec quatre tables dans votre base :

Cours Laravel

Source : https://apcpedagogie.com/les-operations-crud-avec-eloquent-laravel-1-3 · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com