# Laravel : Gestion des modèles, migrations, et seeding

Laravel est un **framework PHP** qui simplifie le développement d'applications web, en particulier pour les interactions avec les bases de données à travers les **modèles**, **migrations**, et **seeders**.

---

## Concepts clés

1. **Modèle (Model)** :
   - **Définition** : C'est une classe PHP qui représente une table de votre base de données.
   - **Rôles** :
     - Gérer les interactions avec une table (exécuter des requêtes SQL comme `SELECT`, `INSERT`, `UPDATE`, `DELETE`).
     - Faciliter l'accès aux données en utilisant des méthodes simples au lieu d'écrire directement du SQL.
2. **Migration** :
   - **Définition** : Une migration est un fichier PHP permettant de créer ou de modifier une table dans la base de données.
   - **Rôles** :
     - Effectuer des opérations DDL (`CREATE`, `ALTER`, `DROP`).
     - Garder une trace des modifications de votre base de données.
3. **Seeder** :
   - **Définition** : Une classe spéciale pour insérer des données fictives ou initiales dans la base de données.
   - **Rôles** :
     - Remplir la base avec des échantillons de données.
     - Faciliter les tests et le développement en créant une base de données utilisable rapidement.

---

## Étapes pratiques : Création de la base de données

### 1. Création de la base de données (nom : `fullstack`)

Créez une base de données dans MySQL ou un autre gestionnaire :

```
CREATE DATABASE fullstack;
```

### 2. Configuration dans `.env`

Dans le fichier `.env` de Laravel, configurez votre base de données :

```
DB_DATABASE=fullstack
DB_USERNAME=your_username
DB_PASSWORD=your_password
```

---

## Avant de commencer : Ajustement dans AppServiceProvider

Pour éviter des erreurs lors de la création des tables, définissez la longueur par défaut des chaînes dans le fichier `app/Providers/AppServiceProvider.php` :

```
use Illuminate\Support\Facades\Schema;

public function boot()
{
    Schema::defaultStringLength(200);
}
```

---

## Création de modèles et migrations

### 3. Création du modèle et de la migration pour `articles`

**Commande pour créer un modèle seul** :

```
php artisan make:model Article
```

**Commande pour créer une migration seule** :

```
php artisan make:migration create_articles_table
```

**Commande combinée pour créer un modèle et une migration** :

```
php artisan make:model Article -m
```

---

## 4. Définir la structure de la table dans la migration

Dans le fichier généré dans `database/migrations` (par exemple `create_articles_table.php`), définissez la structure de la table `articles` :

```
public function up()
{
    Schema::create('articles', function (Blueprint $table) {
        $table->id();
        $table->string('titre'); // VARCHAR
        $table->text('contenu'); // TEXT
        $table->text('image');   // TEXT
        $table->timestamps();    // created_at et updated_at
    });
}
public function down()

{

   Schema::dropIfExists('articles');

 }
```

### 5. Exécuter la migration

## Si vous avez modifié les paramètres de votre fichier .env :

Effacez le cache de configuration Laravel

```
php artisan config:clear
php artisan cache:clear
php artisan config:cache
```

## Pour créer la table articles :

```
php artisan migrate
```

## Pour exécuter une migration spécifique :

```
php artisan migrate --path=/database/migrations/nom_migration.php
```

## Pour réinitialiser toutes les migrations :

```
php artisan migrate:reset
```

---

## Modification d'une table existante

### 6. Modifier la structure de la table `articles`

1. Créez une nouvelle migration :

   ```
   php artisan make:migration modify_articles_table
   ```

2. Ajoutez des modifications dans la méthode `up()` :

   ```
   public function up()
   {
       Schema::table('articles', function ($table) {
           $table->text('image')->nullable()->change(); // Modifier une
   colonne
           // $table->renameColumn('ancien_nom', 'nouveau_nom'); //
   Renommer une colonne
           // $table->text('nouvelle_colonne')->nullable(); // Ajouter
   une colonne
           // $table->dropColumn('colonne_a_supprimer'); // Supprimer
   une colonne
       });
   }
   ```

3. Installez la dépendance nécessaire pour certaines modifications :

   ```
   composer require doctrine/dbal
   ```

4. Exécutez la migration :

   ```
   php artisan migrate
   ```

---

## Ajouter des données : Seeder

### 7. Créer un Seeder

1. Créez un seeder :

   ```
   php artisan make:seed ArticleSeeder
   ```

2. Dans `database/seeders/ArticleSeeder.php`, ajoutez les données aléatoires :

   ```
   use App\Models\Article;
   use Faker\Factory as Faker;

   public function run()
   {
       $faker = Faker::create();

       for ($i = 0; $i < 26; $i++) {
           Article::create([
               'titre' => $faker->sentence(),
               'contenu' => $faker->text(600),
               'image' => $faker->imageUrl()
           ]);
       }
   }
   ```

### 8. Exécuter le Seeder

Pour remplir la table `articles` :

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

---

## Récapitulatif des commandes importantes

1. **Créer un modèle** : `php artisan make:model NomModel`
2. **Créer une migration** : `php artisan make:migration NomMigration`
3. **Créer modèle + migration** : `php artisan make:model NomModel -m`
4. **Exécuter les migrations** : `php artisan migrate`
5. **Créer un seeder** : `php artisan make:seed NomSeeder`
6. **Exécuter un seeder** : `php artisan db:seed --class=NomSeeder`

---

**Annuler la dernière migration :**

php artisan migrate:rollback

**Annuler les n dernières migrations :**

php artisan migrate:rollback --step=n

**Réinitialiser les migrations:**

php artisan migrate:reset

**Supprimer toutes les tables puis relancer toutes les migrations :**

php artisan migrate:fresh

## Conclusion

Grâce à ces outils de Laravel, vous pouvez gérer efficacement vos bases de données sans écrire directement de SQL.

# Laravel Tinker et Eloquent ORM :

Laravel Tinker est un outil très pratique pour tester des interactions avec la base de données en utilisant Eloquent ORM. Voici une exploration plus approfondie des fonctionnalités d'Eloquent avec des exemples pratiques :

---

## Lancer Laravel Tinker

Dans votre terminal, exécutez la commande suivante pour entrer dans l'environnement interactif de Tinker :

```
php artisan tinker
```

## 1. Ajouter plusieurs enregistrements en masse

Eloquent permet d'insérer plusieurs lignes en une seule commande en utilisant la méthode `insert()` :

```
App\Models\User::insert([['name' => 'Alice', 'email' =>
'alice@example.com', 'password' => bcrypt('password123')],

    ['name' => 'Bob', 'email' => 'bob@example.com', 'password' =>
bcrypt('password123')],

]);
```

**Note :** La méthode `insert()` contourne la protection contre les "assignations massives" (`fillable`) et ne déclenche pas les événements Eloquent.

## Configurer les colonnes "fillable" dans le modèle

- La méthode `create()` est protégée contre les "assignations massives". Cela signifie que seules les colonnes définies dans la propriété `$fillable` du modèle peuvent être remplies par des données.
- Exemple :

```
class User extends Model
{
    protected $fillable = ['name', 'email', 'password'];
}
```

- Dans cet exemple, seules les colonnes `name`, `email`, et `password` peuvent être utilisées lors de l'utilisation de `create()`.

---

## 2. Récupérer des enregistrements avec des conditions avancées

Eloquent facilite l'application de conditions complexes :

```
// Récupérer les utilisateurs dont l'email contient 'gmail.com'
$users = User::where('email', 'LIKE', '%gmail.com%')->get();

// Récupérer les utilisateurs créés dans les 7 derniers jours
$recentUsers = User::where('created_at', '>=', now()->subDays(7))->get();

// Récupérer les utilisateurs triés par ordre alphabétique
$sortedUsers = User::orderBy('name', 'asc')->get();
```

---

## 3. Mettre à jour plusieurs enregistrements

Vous pouvez utiliser la méthode `update()` pour modifier plusieurs lignes en une seule commande :

```
// Mettre à jour tous les utilisateurs avec le statut "inactive"
User::where('status', 'active')->update(['status' => 'inactive']);
```

---

## 4. Supprimer plusieurs enregistrements

La méthode `delete()` permet de supprimer plusieurs lignes correspondant à une condition :

```
// Supprimer tous les utilisateurs inactifs
User::where('status', 'inactive')->delete();
```

---

## 5. Méthodes pratiques pour récupérer des données

- **Premier enregistrement correspondant à une condition :**

  ```
  $user = User::where('email', 'john@example.com')->first();
  ```

- **Récupérer un enregistrement ou le créer s'il n'existe pas (`firstOrCreate`) :**

  ```
  $user = User::firstOrCreate(
      ['email' => 'newuser@example.com'],
      ['name' => 'New User', 'password' => bcrypt('password123')]
  );
  ```

- **Mettre à jour ou créer un enregistrement (`updateOrCreate`) :**

  ```
  $user = User::updateOrCreate(
      ['email' => 'existinguser@example.com'],
      ['name' => 'Updated Name', 'password' => bcrypt('newpassword')]
  );
  ```

---

# Équivalence entre requêtes SQL et requêtes Eloquent Laravel

---

Laravel est un framework PHP moderne qui facilite l'interaction avec les bases de données grâce à Eloquent, son ORM (Object Relational Mapper). Eloquent traduit les actions sur des modèles en requêtes SQL de manière intuitive et lisible.

---

## 1. Insertion de données

### SQL :

```
INSERT INTO stagiaire VALUES ('John Doe', '1990-05-15', 'Paris');
```

### Eloquent Laravel :

```
Stagiaire::create([
    'nom' => 'John Doe',
    'date_naissance' => '1990-05-15',
    'ville' => 'Paris'
]);
```

---

## 2. Mise à jour de données

### SQL :

```
UPDATE stagiaire SET ville = 'Lyon' WHERE id = 1;
```

### Eloquent Laravel :

### Mise à jour d'un enregistrement spécifique :

```
$stagiaire = Stagiaire::find(1);
$stagiaire->ville = 'Lyon';
$stagiaire->save();
```

### Mise à jour de plusieurs enregistrements avec condition :

```
Stagiaire::where('ville', 'Paris')->update(['ville' => 'Lyon']);
```

---

## 3. Suppression de données

### SQL :

```
DELETE FROM stagiaire WHERE id = 1;
```

### Eloquent Laravel :

### Suppression d'un enregistrement spécifique :

```
$stagiaire = Stagiaire::find(1);
$stagiaire->delete();
```

### Suppression multiple avec condition :

```
Stagiaire::where('ville', 'Paris')->delete();
```

---

## 4. Sélection de données

### SQL :

```
SELECT * FROM stagiaire;
```

### Eloquent Laravel :

```
$stagiaires = Stagiaire::all();
```

### Avec condition (WHERE) :

```
$stagiaires = Stagiaire::where('ville', 'Paris')->get();
```

### Avec plusieurs conditions (AND/OR) :

```
// AND
$stagiaires = Stagiaire::where('ville', 'Paris')
                       ->where('date_naissance', '>=', '2000-01-01')
                       ->get();

// OR
$stagiaires = Stagiaire::where('ville', 'Paris')
                       ->orWhere('ville', 'Lyon')
                       ->get();
```

---

## 5. Pagination

Pour afficher les données par pages :

```
$clients = Client::paginate(10);
```

### Dans une vue Blade :

```
@foreach ($clients as $client)
    <p>{{ $client->nom }}</p>
@endforeach

{{ $clients->links() }}
```

Laravel génère automatiquement des liens pour naviguer entre les pages.

---

## 6. Cas pratiques avec tri et limites

### SQL :

```
SELECT * FROM stagiaire WHERE ville = 'Paris' ORDER BY nom ASC LIMIT 10;
```

### Eloquent Laravel :

```
$stagiaires = Stagiaire::where('ville', 'Paris')
                       ->orderBy('nom', 'asc')
                       ->limit(10)
                       ->get();
```

---

## 7. Exemple avancé de mise à jour

### SQL :

```
UPDATE commandes SET statut = 'en attente de paiement'
WHERE montant > 1000 AND date_creation < '2023-01-01';
```

### Eloquent Laravel :

```
Order::where('montant', '>', 1000)
     ->where('created_at', '<', '2023-01-01')
     ->update(['status' => 'en attente de paiement']);
```

---

## Résumé et bonnes pratiques :

1. **Lisibilité :** Eloquent privilégie une syntaxe claire et expressive.
2. **Productivité :** Moins de code pour des requêtes SQL complexes.
3. **Sécurité :** Protection contre les injections SQL.
4. **Structure MVC :** Favorise une séparation nette entre la logique métier et la couche base de données.

**Atelier 5 – Travail à faire**

Créer une base de données MySQL sur un serveur distant en utilisant AlwaysData.

Créer un utilisateur MySQL et lui attribuer les droits nécessaires sur la base.

Configurer la connexion à la base de données dans le fichier .env du projet Laravel.

Créer une migration pour la table articles.

Exécuter les migrations afin de créer la table dans la base de données distante.

Créer un seeder pour insérer des données de test dans la table articles.

Exécuter le seeder pour remplir la base de données.

Rédiger un rapport contenant les captures d’écran des étapes réalisées.