Gestion des bases de données dans Laravel :

1. Migrations

Les migrations permettent de versionner la structure de la base de données et de faciliter la gestion des modifications de schéma.

- **Créer une migration** :

  ```
  php artisan make:migration create_nom_de_table_table
  ```

- **Exécuter les migrations** (applique les changements sur la base de données) :

  ```
  php artisan migrate
  ```

- **Annuler la dernière migration** :

  ```
  php artisan migrate:rollback
  ```

- **Réinitialiser toutes les migrations** (annule toutes les migrations et les réexécute) :

  ```
  php artisan migrate:reset
  ```

- **Refaire toutes les migrations** (effectue un `rollback` suivi d'un `migrate`) :

  ```
  php artisan migrate:refresh
  ```

- **Exécuter des migrations spécifiques** :

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

- **Vérifier l'état des migrations** (si elles ont été appliquées ou non) :

  ```
  php artisan migrate:status
  ```

- **Créer une migration pour modifier une table existante** :

  ```
  php artisan make:migration add_column_to_nom_de_table_table --
  table=nom_de_table
  ```

2. Seeders

Les seeders sont utilisés pour insérer des données initiales dans la base de données.

- **Créer un seeder** :

  ```
  php artisan make:seeder NomDuSeeder
  ```

- **Exécuter les seeders** (insère les données dans la base) :

  ```
  php artisan db:seed
  ```

- **Exécuter un seeder spécifique** :

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

- **Réexécuter les migrations et les seeders** :

  ```
  php artisan migrate:refresh --seed
  ```

3. Factories

Les factories sont utilisées pour générer des données factices pour tester l'application.

- **Créer une factory** :

  ```
  php artisan make:factory NomDeLaFactory
  ```

- **Utiliser une factory dans un seeder** pour générer des données factices :

  ```
  \App\Models\NomDuModel::factory()->count(50)->create();
  ```

4. Gestion de la base de données

- **Vider la base de données (toutes les tables)** :

  ```
  php artisan db:wipe
  ```

- **Lancer une requête SQL personnalisée** dans un seeder ou un contrôleur :

  ```
  DB::statement('SQL_QUERY');
  ```

5. Base de données et migrations avec Artisan

- **Exporter la structure de la base de données dans un fichier SQL** : Laravel ne dispose pas d'une commande native pour l'exportation de la base de données, mais vous pouvez utiliser des outils externes ou des packages comme **`spatie/laravel-db-snapshots`.**
- **Importer une base de données** (via un dump SQL ou un autre outil externe).

6. Base de données en environnement de développement

- **Recréer la base de données et les migrations** : Cela implique de supprimer et de réexécuter toutes les migrations.

  ```
  php artisan migrate:fresh
  ```

7. Utiliser les Transactions de Base de Données

Laravel fournit un support pour les transactions via la méthode `DB::transaction` :

- **Exécuter une transaction** :

```
DB::transaction(function () {
    // Vos opérations sur la base de données
});
```

8. Affichage des requêtes SQL

- **Afficher les requêtes SQL exécutées par Laravel** (utile pour le débogage) :

```
DB::listen(function ($query) {
    var_dump($query->sql);
});
```

9. Manipulation des données avec Eloquent (Modèles)

- **Récupérer toutes les entrées d'une table** :

```
$model = NomDuModel::all();
```

- **Récupérer une entrée spécifique par son ID** :

```
$model = NomDuModel::find($id);
```

- **Sauvegarder une entrée dans la base de données** :

```
$model = new NomDuModel;
$model->champ = 'valeur';
$model->save();
```

- **Mettre à jour une entrée existante** :

```
$model = NomDuModel::find($id);
$model->champ = 'nouvelle valeur';
$model->save();
```

- **Supprimer une entrée** :

```
$model = NomDuModel::find($id);
$model->delete();
```

10. Commandes utiles pour le débogage

- **Afficher le dernier query exécuté** (avec `DB::getQueryLog()` si activé) :

```
DB::enableQueryLog();
// Exécuter la requête
dd(DB::getQueryLog());
```