Gestion de Migration

Introduction

Une migration permet de créer et de mettre à jour un schéma de base de données. Autrement dit, vous pouvez créer des tables, des colonnes dans ces tables, en supprimer, créer des index… Tout ce qui concerne la maintenance de vos tables peut être pris en charge par cet outil. Vous avez ainsi un suivi de vos modifications.

La migration Laravel est comme un outil de gestion de version de DATABASE.

La configuration de la base

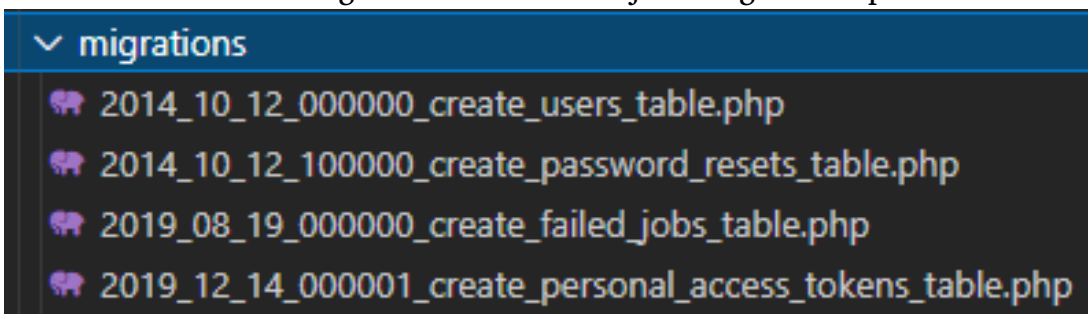
Laravel permet de gérer les bases de type **MySQL**,**PostgreSql**,**SQLite** et **SQLServer**.

Il faut indiquer où se trouve votre base, son nom, le nom de l'utilisateur, le mot de passe dans le fichier de configuration **.env** :

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

Installation

Le dossier database/migrations contient déjà 4 migrations présentes :



create_users_table : c'est une migration de base pour créer une table des utilisateurs,

create_password_resets_table : c'est une migration liée à la précédente qui permet de gérer le renouvellement des mots de passe en toute sécurité,

create_failed_jobs_table : une migration qui concerne les queues,

create_personal_access_tokens_table : concerne les api.

Exécution de migrations

supposons qu'on a nos migrations bien créées (voir la suite), on aura besoin, alors, de l'une des commandes suivantes :

lancer les migrations	**php artisan migrate**
Annuler la **dernière** migration	**php artisan migrate:rollback**
Annuler la migration jusqu'à **n** étapes en commençant par la plus récente.	**php artisan migrate:rollback -step=n**
Annuler **toutes** les opérations	**php artisan migrate:reset**
Afficher l'état des migrations	**php artisan migrate:status**
recrée efficacement l'intégralité de votre base de données	**php artisan migrate:refresh**
Annuler et migrer à nouveau un **nombre limité** de migrations.	**php artisan migrate:refresh -step=n**
Supprimera toutes les tables de la base de données, puis exécutera la commande migrate	**php artisan migrate:fresh** NB : refresh=(rollback all + migrate) fresh=(drop all + migrate)

Créer une migration

Pour générer une migration, vous devez exécuter une commande

```
php artisan make:migration create_stagiaires_table
```

Cela générera un fichier dans le dossier **database\migrations**

Le fichier consiste en une nouvelle classe étendant la classe de migration de LARAVEL. On dispose dans cette classe de deux fonctions :

up : ici on a le code de création de la table et de ses colonnes (le code qui sera exécuté lorsqu'on lance la migration)

down : ici on a le code de suppression de la table (le code qui sera exécuté lorsqu'on annule la migration)

```php
public function up()
{
    Schema::create('stagiaires', function (Blueprint $table) {
      $table->id();
      $table->string('nom');
      $table->string('prenom', 50);
      / /$table->boolean('etat')->default(0);
      $table->timestamps();
    });
  }
```

```php
public function down()  {    Schema::dropIfExists('stagiaires');  }
```

Exercice :

1- créer une migration qui inclut le code de création d'une nouvelle table 'Examen' sachant qu'un examen est caractèrisé par : un codeExam, un libelle, une durée et une date

2- lancer la migration et vérifier que la table est bien ajoutée à votre base de données.

Modification du schéma d'une base de données

La méthode ***table*** sur la façade ***Schema*** peut être utilisée pour mettre à jour des tables existantes. Comme la méthode ***create***, la méthode ***table*** accepte deux arguments : le nom de la table et une fermeture qui reçoit une instance Blueprint que vous pouvez utiliser pour ajouter des colonnes ou des index à la table :

1- ajouter une nouvelle colonne à un table existante :

```php
Schema::table('stagiaires', function (Blueprint $table) {
     $table->integer('age')->after(('prenom'));
    });
```

Remarque 1 : il est préférable de créer une migration spécifique à l'opération voulue et éviter d'enchaîner les opérations dans la même migration.

Pour cela, pour ajouter la nouvelle colonne 'age', on crée la migration 'add_age_to_stagiaires', puis, dans la fonctuion up , on insére le code ci-dessus.

Remarque2 : différents types sont disponibles (voir ici : https://laravel.com/docs/9.x/migrations#available-column-types)

Remarque 3 : Le tableau suivant contient tous les modificateurs de colonne disponibles. Cette liste n'inclut pas les modificateurs d'index :

Modificateur	Description
->after('column')	Placer la colonne "après" une autre colonne (MySQL).
->autoIncrement()	Définit les colonnes INTEGER comme auto-incrémentée
>first()	Place la colonne "first" dans la table (MySQL).
->nullable()	Autorise l'insertion de valeurs NULL dans la colonne
->unsigned()	Définit les colonnes INTEGER comme UNSIGNED (MySQL).
->useCurrent()	Définit les colonnes TIMESTAMP pour utiliser CURRENT_TIMESTAMP comme valeur par défaut.
->useCurrentOnUpdate()	Définit les colonnes TIMESTAMP pour utiliser CURRENT_TIMESTAMP lorsqu'un enregistrement est mis à jour.
….	

2- Ajouter une contrainte :

De la même façon, on peut ajouter des contraintes sur les colonnes déjà définis via la méthode **table** et la **méthode change** :

```
Schema::table('stagiaires', function (Blueprint $table) {
$table->string('fil')->default('fullstack')->change();});
}
```

Créer des clés étrangères à l'aide de migrations

soient le schéma relationnel suivant :

stagiaires(ids,nom, prenom)

notations(idn,#ids,note)

La table stagiaire

```
public function up() {
        Schema::create('stagiaires', function (Blueprint $table) {
        $table->id();
        $table->string('nom')->unique();
        $table->string('prenom')->nullable();
        $table->timestamps();
        });     }
```

La table notation (Méthode1) :

```
public function up()
{
Schema::disableForeignKeyConstraints();
Schema::create('notations', function (Blueprint $table) {
  $table->id();
$table->foreign('stagiaire_id')
        ->references('id')
        ->on('stagiaires')
        ->onDelete('restrict')
        ->onUpdate('restrict');
});
}
```

La table notations (2 -ème façon) *a partir de laravel 8 et plus*

```
public function up()
{
    Schema::disableForeignKeyConstraints();
    Schema::create('notations', function (Blueprint $table) {
        ...
 $table->foreignId('stagiaire_id')->constrained()
//constrained('stagiaires')
            ->onUpdate('restrict')
            ->onDelete('restrict');
});
}
```

Dans la table notations on déclare une clé étrangère nommée **stagiaire_id** qui référence la colonne **id** dans la table **stagiaires.**

La méthode **foreignId** crée une colonne de type **UNSIGNED BIGINT.** La méthode **constrained** utilise les conventions de Laravel pour 0déterminer le nom de la table référencée, ici c'est **stagiaires.**

En cas de suppression (**onDelete**) ou de modification (**onUpdate**) on a une restriction (**restrict**).

Une autre possibilité est **cascade** à la place de **restrict.** Dans ce cas si vous supprimez un stagiaire ça supprimera en cascade les notations de ce stagiaire.

Les migrations sont effectuées dans l'ordre alphabétique, ce qui peut générer un problème avec les clés étrangères. Si la table référencée est créée après. C'est pour cette raison qu'on a ajouté cette ligne dans la migration :

```
Schema::disableForeignKeyConstraints();
```

On désactive temporairement le contrôle référentiel le temps de créer les tables.

2- Renommer une colonne:

Pour renommer une colonne, vous pouvez utiliser la méthode **renameColumn** fournie par le constructeur de schéma

```
Schema::table('stagiaires', function (Blueprint $table) {
$table->renameColumn('from', 'to');
}
```

2- Supprimer une colonne:

Pour supprimer une colonne, vous pouvez utiliser la méthode **dropColumn** fournie par le constructeur de schéma

```
Schema::table('users', function (Blueprint $table) {
$table->dropColumn('age');
});
```

Création d'index

Pour créer l'index, nous pouvons enchaîner la méthode **unique** sur la définition de colonne :

```
Schema::table('stagiaires', function (Blueprint $table) {
            $table->integer('age')->unique());
        });
```

Remarque 1 :
Vous pouvez également créer l'index après avoir défini la colonne. Pour ce faire, vous devez appeler la méthode **unique** . Cette méthode accepte le nom de la colonne qui doit recevoir un index unique : **$table→unique('email');**
Remarque 2 :
Vous pouvez même passer un tableau de colonnes à une méthode **index** pour créer un index composé (ou composite) : `$table->index(['account_id', 'created_at']);`

Remarque 3 : Lors de la création d'un index, Laravel générera automatiquement un nom d'index basé sur la table, les noms de colonne et le type d'index, mais vous pouvez passer un deuxième argument à la méthode pour spécifier vous-même le nom de l'index : **$table->unique('email', 'unique_email');**

Laravel pris en charge différents types d'index.Ainsi, on a des méthodes pour créer chaque type d'index. Chaque méthode d'index accepte un deuxième argument facultatif pour spécifier le nom de l'index. S'il est omis, le nom sera dérivé des noms de la table et des colonnes utilisées pour l'index, ainsi que du type d'index (exemple :

`users_id_primary`) . Chacune des méthodes d'indexation disponibles est décrite dans le tableau ci-dessous :

Command	Description
`$table->primary('id');`	Adds a primary key.
`$table->primary(['id', 'parent_id']);`	Adds composite keys.
`$table->unique('email');`	Adds a unique index.
`$table->index('state');`	Adds an index.
`$table->fullText('body');`	Adds a full text index (MySQL/PostgreSQL).

- pour renommer un index, utilisez la méthode ***renameIndex*** : **`$table->renameIndex('from', 'to')`**
- pour supprimer un index, on peut utiliser l'une des méthodes : dropIndex(nameindex), dropUnique(nameindex),dropPrimary(nameindex)…

TP :

1. Créer un nouveau projet Laravel : TPMigrationBD
2. Configurer le fichier .env pour se connecter à la base de données : testsLaravel que vous devrez créer manuellement via l'interface phpmyadmin
3. Créer une migration de la table Livres avec les colonnes suivantes : (isbn : entier qui représente l'id auto-incrémenté, titre : chaine de 50 caractères en maximum, auteur : chaine de caractères, prix : nombre entier positif)
4. créer une migration qui concerne l'ajout d'une nouvelle colonne 'nombre de pages' : entier positif
5. Lancer les migrations existantes.Puis, vérifier votre bd.
6. annuler la dernière migration.
7. ajouter, manuellement dans le phpmyadmin, une table auteur avec les champs : codea, nom, prenom,daten, nationnalité (n'ajouter pas la contrainte primary key)
8. créer une migration qui modifie le type de la colonne 'auteur' dans la table 'livre' en un entier positif.
9. créer une migration pour définir la colonne 'auteur' de la table 'livre' comme clé étrangère
10. créer un index primary sur le champ codea (migration)