Banco de Dados: Migrations

Introdução

Migrations são como controle de versão para seu banco de dados, permitindo uma equipe de modificar e compartilhar o schema da aplicação facilmente.

O Laravel `Schema` facade fornece um suporte agnóstico para criação e manipulação de tabelas. Ele compartilha o mesmo modelo, API fluente em todos os sitemas de banco de dados suportados pelo Laravel.

Gerando Migrations

Para criar uma migration, use o `make:migration` Artisan command:

```
php artisan make:migration create_users_table
```

A nova migration será colocada em seu diretório `database/migrations` . Cada arquivo migration contém um timestamp que permite o Laravel determinar a ordem em que serão executadas.

As opções `--table` e `--create` também podem ser utilizados para indicar o nome da tabela e se a migration irá criar uma nova tabela. Estas opções simplesmente irão pré-preencher o arquivo da sua migration:

```
php artisan make:migration add_votes_to_users_table --table=users

php artisan make:migration create_users_table --create=users
```

Estrutura da Migration

A Classe migration contém dois métodos: `up` e `down`. O método `up` é usado para adicionar novas tabelas, colunas, ou índices para seu banco de dados, enquanto que o método `down` deve simplesmente inverter as operações realizadas pelo médodo `up`.

Dentro de ambos os métodos você pode utilizar o schema do Laravel para criar e modificar tabelas. Para saber mais sobre todos os métodos disponíveis no construtor Schema, verifique essa documentação. Por exemplo, vamos olhar para uma migration que cria a tabela de `flights`:

```
<?php

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

class CreateFlightsTable extends Migration
{
        /**
         * Run the migrations.
         *
         * @return void
         */
        public function up()
        {
                Schema::create('flights', function (Blueprint $table) {
                        $table->increments('id');
                        $table->string('name');
                        $table->string('airline');
                        $table->timestamps();
                });
        }

        /**
         * Reverse the migrations.
         *
         * @return void
         */
        public function down()
        {
                Schema::drop('flights');
        }
}
```

Executando Migrations

Para executar todas as migrations pendentes da sua aplicação, use o comando artisan `migrate`. Se você estiver usando uma máquina virtual Homestead, você deve executar esse comando dentro de sua VM:

```
php artisan migrate
```

Se você receber um erro "class not found" ao executar as migrations, tente executar o comando `composer dump-autoload` e execute novamente o comando `migrate`

Forçando Migrations Para Rodar em Produção

Algumas operações de migration são destrutivas, que significa que eles podem causar a perda de dados. A fim de protegê-lo de executar esses comandos no banco de dados em produção, você será solicitado para confirmação antes desses comandos serem executados. Para forçar os comandos e executar sem um prompt, use o sinalizador `--force`:

```
php artisan migrate --force
```

Revertendo Migrations

Para reverter a última "operação" da migration, você pode usar o comando `rollback`. Note que esse comando rever o último "lote" de migrations que foram executadas, o que pode incluir vários arquivos:

```
php artisan migrate:rollback
```

O comando `migrate:reset` irá reverter todas as suas migrations de sua aplicação:

```
php artisan migrate:reset
```

Reverter / Migrar um comando individual

O comando `migrate:refresh` irá primeiro recuperar todas as suas migrations de banco de dados, e em seguida executar o comando `migrate`. Este comando efetivamente recria toda a sua base de dados:

```
php artisan migrate:refresh

php artisan migrate:refresh --seed
```

Escrevendo Migrations

Criando Tabelas

Para criar uma nova tabela de banco de dados, use o método `create` no `Schema` facade. O método `create` aceita dois argumentos. O primeiro é o nome da tabela, enquanto o segundo é uma `Closure` que recebe um objeto `Blueprint` para definir uma nova tabela:

```
Schema::create('users', function ($table) {
        $table->increments('id');
});
```

Claro que, ao criar uma tabela, você pode usar qualquer um dos métodos de coluna para definir as colunas da tabela.

Verificação Da Tabela / Existência Da Coluna

Você pode facilmente verificar a existência de uma tabela ou coluna usando os métodos `hasTable` e `hasColumn`:

```
if (Schema::hasTable('users')) {
        //
}

if (Schema::hasColumn('users', 'email')) {
        //
}
```

Conexão & Storage Engine

Se você quiser executar uma operação um uma conexão de banco de dados que não é sua conexão padrão, use o método `connection`:

```
Schema::connection('foo')->create('users', function ($table) {
        $table->increments('id');
});
```

Para definir o storage engine para uma tabela, defina a propriedade `engine` no construtor schema:

```
Schema::create('users', function ($table) {
        $table->engine = 'InnoDB';

        $table->increments('id');
});
```

Renomeando / Deletando Tabelas

Para renomear uma tabela de banco de dados existente, use o método `rename`:

```
Schema::rename($from, $to);
```

Para deletar uma tabela existente, você pode usar os métodos `drop` ou `dropIfExists`:

```
Schema::drop('users');

Schema::dropIfExists('users');
```

Criando Colunas

Para atualizar uma tabela existente, vamos usar o método `table` no `Schema` facade. Como o método `create`, o método `table` também aceita dois argumentos: o nome da tabela e uma `Closure` que recebe uma instância do `Blueprint` que podemos usar para adicionar colunas à tabela:

```
Schema::table('users', function ($table) {
        $table->string('email');
});
```

Tipos de Colunas Disponíveis:

Claro, o construtor schema contém uma variedade de tipos de colunas que podem ser usados para construir suas tabelas:

Comando	Descrição
`$table->bigIncrements('id');`	Incrementando ID usando um "big integer" equivalente.
`$table->bigInteger('votes');`	BIGINT equivalente para o banco de dados.
`$table->binary('data');`	BLOB equivalente para o banco de dados.
`$table->boolean('confirmed');`	BOOLEAN equivalente para o banco de dados.
`$table->char('name', 4);`	CHAR com um comprimento equivalente.
`$table->date('created_at');`	DATE equivalente para o banco de dados.
`$table->dateTime('created_at');`	DATETIME equivalente para o banco de dados.
`$table->decimal('amount', 5, 2);`	DECIMAL equivalente com uma precisão e escala.
`$table->double('column', 15, 8);`	DOUBLE equivalente com precisão, 15 dígitos no total e 8 após o ponto decimal.
`$table->enum('choices', ['foo', 'bar']);`	ENUM equivalente para o banco de dados.
`$table->float('amount');`	FLOAT equivalente para o banco de dados.
`$table->increments('id');`	Incrementando ID para o banco de dados (chave primaria).
`$table->integer('votes');`	INTEGER equivalente para o banco de dados.
`$table->json('options');`	JSON equivalente para o banco de dados.
`$table->jsonb('options');`	JSONB equivalente para o banco de dados.
`$table->longText('description');`	LONGTEXT equivalente para o banco de dados.
`$table->mediumInteger('numbers');`	MEDIUMINT equivalente para o banco de dados.
`$table->mediumText('description');`	MEDIUMTEXT equivalente para o banco de dados.
`$table->morphs('taggable');`	Adiciona um INTEGER `taggable_id` e uma STRING `taggable_type`.
`$table->nullableTimestamps();`	Mesmo como `timestamps()`, exceto permite NULLs.
`$table->rememberToken();`	Adiciona um `remember_token` VARCHAR(100) NULL.
`$table->smallInteger('votes');`	SMALLINT equivalente para o banco de dados.
`$table->softDeletes();`	Adiciona uma coluna `deleted_at` para soft deletes.
`$table->string('email');`	VARCHAR coluna equivalente.

Comando	Descrição
`$table->string('name', 100);`	VARCHAR com um comprimento equivalente.
`$table->text('description');`	TEXT equivalente para o banco de dados.
`$table->time('sunrise');`	TIME equivalente para o banco de dados.
`$table->tinyInteger('numbers');`	TINYINT equivalente para o banco de dados.
`$table->timestamp('added_on');`	TIMESTAMP equivalente para o banco de dados.
`$table->timestamps();`	Adiciona uma coluna `created_at` e `updated_at`.

Modificadores de Coluna

Além dos tipos de colunas listados acima, existem vários outros "modificadores", que você pode usar ao adicionar a coluna. Por exemplo, para fazer a coluna "**null**able", você pode usar o método `nullable`:

```
Schema::table('users', function ($table) {
        $table->string('email')->nullable();
});
```

Abaixo está uma lista de todos os modificadores de coluna disóníveis. Está lista não inclui os modificadores do índice:

Modificador	Descrição
`->after('column')`	Coloque uma coluna "depois(after)" de outra coluna (Apenas MySQL)
`->nullable()`	Permitir valores **NULL** para serem inseridos na coluna
`->default($value)`	Define um valor "default" para a coluna
`->unsigned()`	define `integer` para colunas `UNSIGNED`

Modificando Colunas

Pré-requisitos

Antes de modificar uma coluna, certifique-se de adicionar a dependência `doctrine/dbal` para o seu arquivo `composer.json`. A biblioteca Doctrine DBAL é utilizado para determinar o estado atual da coluna e criar as consultas SQL necessários para fazer as alterações específicas à coluna.

Atualizando Atributos de Coluna

O método `change` permite que você modifique uma coluna existente para um novo tipo, ou modificar atributos da coluna. Por exemplo, você pode desejar aumentar o tamanho de uma coluna string. Para ver o método `change` em ação, vamos aumentar o tamanho da coluna `name` de 25 para 50:

```
Schema::table('users', function ($table) {
        $table->string('name', 50)->change();
});
```

Nós também poderiamos modificar a coluna para ser **null**able:

```
Schema::table('users', function ($table) {
        $table->string('name', 50)->nullable()->change();
});
```

Renomeando Colunas

Para renomear uma coluna, você pode usar o método `renameColumn` no Schema. Antes de renomear uma coluna, certifique-se de adicionar a dependência `doctrine/dbal` para o seu arquivo `composer.json`.

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

> **Nota:** Não é suportado enomear colunas de uma tabela com `enum`.

Deletando Colunas

Para deletar uma coluna, use o método `dropColumn` do Schema:

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

Você pode deletar várias colunas da tabela passando um array com os nomes das colunas com o método `dropColumn`:

```
Schema::table('users', function ($table) {
        $table->dropColumn(['votes', 'avatar', 'location']);
});
```

> **Note:** Antes de deletar as colunas de um banco de dados SQLite, você precisa adicionar a dependência `doctrine/dbal` para o seu arquivo `composer.json` e executar o comando `composer update` em seu terminal para instalar a biblioteca.

Criando Índices

O construtor schema suporta diversos tipos de índices. Primeiro, vamos olhar um exemplo que especifica que o valor da coluna deve ser único. Para criar o índice, podemos simplesmenteo método `unique` para a definição da coluna:

```
$table->string('email')->unique();
```

Alternativamente, você pode criar o índice após a definição da coluna. Por exemplo:

```
$table->unique('email');
```

Você pode até passar uma matriz de colunas para o método criar um índice composto:

```
$table->index(['account_id', 'created_at']);
```

Tipos de índice disponíveis

Comando	Descrição
`$table->primary('id');`	Adiciona uma chave primaria.
`$table->primary(['first', 'last']);`	Adiciona chaves compostas.
`$table->unique('email');`	Adiciona um índice único.
`$table->index('state');`	Adiciona um índece básico.

Deletando Índices

Para excluir um índice, você deve especificar o nome do índice. Por padrão, o Laravel atribui automaticamente um nome razoável para os índices. Simplesmente concatena com o nome da tabela, os nomes da coluna no índice e o tipo de índice. Aqui estão alguns exemplos.

Comando	Descrição
`$table->dropPrimary('users_id_primary');`	Deleta a chave primária da tabela "users".
`$table->dropUnique('users_email_unique');`	Delete o índice único da tabela "users".
`$table->dropIndex('geo_state_index');`	Deleta o índice básico da tabela "geo".

Restrições de Chave Estrangeira

O Laravel também fornece suporte para a criação de 'constraints', que são usados para forçar a integridade referencial no nível de banco de dados. Por exemplo, vamos definir uma coluna `user_id` na tabela `posts` que faz referência a coluna `id` da tabela `users` :

```
Schema::table('posts', function ($table) {
        $table->integer('user_id')->unsigned();

        $table->foreign('user_id')->references('id')->on('users');
});
```

Você também pode especificar a ação desejada para "on delete" and "on update" propriedades da restrição(constraint):

```
$table->foreign('user_id')
          ->references('id')->on('users')
          ->onDelete('cascade');
```

Para excluir uma chave estrangeira, você pode usar o método `dropForeign` . Restrições de chaves estrangeiras usam a mesma convenção de nomenclatura como índices. Então vamos concatenar o nome da tabela e as colunas da restrição(constraint), em seguida com "_foreign":

```
$table->dropForeign('posts_user_id_foreign');
```