SUMÁRIO

1. FUNDAMENTOS DO LARAVEL: BANCO DE DADOS

Quase todos os aplicativos da web modernos interagem com um banco de dados. O Laravel torna a interação com bancos de dados extremamente simples através de uma variedade de bancos de dados suportados, um criador de consultas fluente e o Eloquent ORM.

1.1. ARQUIVO DE CONFIGURAÇÃO

A configuração dos serviços de banco de dados do Laravel está localizada no arquivo de configuração **config/database.php**.

Os valores sigilosos existentes neste arquivo, como usuário e senha do banco de dados, entre outros tipos de dados, **são orientados pelos valores das variáveis de ambiente** setadas no arquivo **/.env**.

2. EXECUTANDO QUERIES NO LARAVEL

2.1. FORMA CONVENCIONAL: CLASSE PDO

2.2. QUERY BUILDER

O Query Builder do Laravel fornece, a partir da classe **Illuminate\Support\Facades\DB**, uma interface conveniente e fluente para criar e executar consultas de banco de dados, e **sem precisar de Models padronizados**.

Obviamente, para respeitar a arquitetura MVC, a comunicação com o banco deve ser realizada em um Model, mas, um tradicional, com atributos e métodos quaisquer adaptados à aplicação. Já com Eloquent ORM, que será abordado posteriormente, os Models, para funcionarem, precisam seguir alguns padrões de codificação.

O construtor de consultas Laravel usa ligação de parâmetros PDO para proteger sua aplicação contra ataques de injeção de SQL. Não há necessidade de limpar ou higienizar strings passadas para o construtor de consultas como ligações de consulta.

```
$data = DB::table('users')->where('name', 'John');
```

2.3. ELOQUENT ORM E OS MODELS

O Eloquent ORM é uma outra forma de executar comandos SQL, e a forma mais comum para essa tarefa dentro do ambiente Laravel.

Ao usar o Eloquent, cada tabela de banco de dados deve possuir um "Model" correspondente que é usado para interagir com ela. Além de recuperar registros da tabela do banco de dados, o Eloquent também permite inserir, atualizar e excluir registros da tabela.

3. MAIS SOBRE OS MODELS

Pois bem: os models são os arquivos que fazem parte da camada de dados, e que são invocados a partir dos controladores. Eles realizam a comunicação direta com o banco de dados, e fazem isso por meio do Eloquent ORM.

Os modelos geralmente residem no diretório app\Models e estendem a classe Illuminate\Database\Eloquent\Model.

3.1. COMO CRIAR UM MODEL

Para criar um Model, deve ser utilizado o comando make:model do console Artisan. Veja abaixo:

Você pode usar o make:model comando Artisan para gerar um novo modelo:

php artisan make:model NomeModel

3.2. ORGANIZAÇÃO DE UM MODEL

Os modelos gerados pelo comando make:model serão colocados no diretório app/Models. A organização inicial de um Model é essa:

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class NomeModel extends Model

{ // }
```

3.2.1. DEFININDO A TABELA DO MODEL

É preciso informar ao Eloquent qual tabela do banco de dados o Model gerado representa. Se não for informado nenhum, ele irá considerar uma tabela com o mesmo nome do Model. Assim, se a tabela correspondente não seguir essa convenção, isto deve ser feito:

```
class NomeModel extends Model

{

  Protected $table = “nome_tabela”; // Tabela do Model
```

```
}
```

3.2.2. DEFININDO A CHAVE PRIMÁRIA DA TABELA DO MODEL

O Eloquent também assumirá que cada tabela de banco de dados correspondente do modelo possui uma coluna de chave primária chamada “id”. Assim, se não for esse o nome da chave primária, o Eloquent deverá ser informado.

```
class NomeModel extends Model
{
    protected $table = “nome_tabela”; // Tabela do Model
    protected $primaryKey = “nome_da_coluna”; // Nome da PK
}
```

Além disso, o Eloquent assume que a chave primária é um valor inteiro crescente, o que significa que o Eloquent converterá automaticamente a chave primária em um inteiro. Se desejar usar uma chave primária não incremental ou não numérica, faça isso:

```
class NomeModel extends Model
{
    protected $table = “nome_tabela”; // Tabela do Model
    protected $primaryKey = “nome_da_coluna”; // Nome da PK
    public $incrementing = false; // PK com auto_increment
}
```

Existem diversos outros valores de configuração que o Eloquent assume, e que devem ser alterados caso não sejam verdadeiros. Para saber mais clique aqui.

4. SOBRE MIGRATIONS

As migrações são como o controle de versão do seu banco de dados, permitindo que sua equipe defina e compartilhe a definição do esquema do banco de dados do aplicativo.

O Facade “Schema” do Laravel fornece suporte para criar e manipular tabelas em todos os sistemas de banco de dados suportados pelo Laravel. Normalmente, as migrações usarão essa Facade para criar e modificar as tabelas e colunas do banco de dados.

4.1. CRIANDO UMA MIGRATION E COMANDOS FREQUENTES

Você pode usar o comando Artisan **make:migration** para gerar uma migração de banco de dados. A nova migração será colocada no diretório database/migrations.

php artisan make:migration create_nome**_table**

O Laravel usará o nome do migrate para tentar adivinhar o nome da tabela. Se o Laravel for capaz de determinar o nome da tabela a partir do nome do migrate, ele irá pré-preencher o arquivo com os dados da tabela especificada. Caso contrário, você pode simplesmente especificar a tabela no arquivo de migração manualmente.

Mas esse comando não cria, de imediato, a tabela, e sim apenas o migrate dela, isto é, o arquivo que permitirá que seja criada ou excluída no próprio ambiente Laravel, e no nível do código. **Para executar a migração criada**, neste caso a criação da tabela, esse comando é necessário:

php artisan migrate

Para ver as migrações executadas, e as não executadas, basta digitar esse outro comando:

php artisan migrate:status

Para reverter a última ação realizada no âmbito das migrações, basta digitar:

php artisan migrate:rollback

Para recriar uma tabela sem perder os seus dados é preciso utilizar um comando diferente de php artisan migrate. Se esse for usado, a tabela, sim, será criada novamente, mas todos os seus dados serão excluídos. Para que isso não ocorra, basta executar esse comando:

php artisan migrate:refresh

4.2. O CÓDIGO DE UMA MIGRATION

Existem, sempre, dois métodos em uma migração: o **up()**, que realiza as ações, e o **down()** que reverte as ações. **Mas com quais comandos exatamente eles são executados?** O up() será executado com o comando migrate ou migrate:refresh, e o down() será executado com o comando rollback.

Como será visto no próximo tópico, existem tipos de migrações, e na que já foi mostrada, do tipo “create”, o método up() terá a rotina para criar a tabela e as suas colunas, enquanto e o método down() terá a rotina, por padrão, para excluir a tabela, se existir.

4.3. MÉTODO CREATE E TABLE DA CLASSE SCHEMA

Existem tipos dois tipos de migração, e o fator dessa variação é causado pela forma como o comando é escrito. A **variação que ocorre é o método utiliza pelo Facade Schema**.

Se o comando começar com create_, o Laravel irá criar uma migration com o Schema::create() no método up(), e Schema::dropIfExists() no método down().

Se o comando não começar com create_, independente do resto, servirá para manipular uma tabela que já existe. Neste caso, o Laravel irá criar uma migration com o Schema::table tanto no método up() quanto no método down().

Ou seja: o método “create”, da Facade Schema, serve para criar tabelas novas, e o método “table” para modificar uma já existente.