BANCO DE DADOS COM LARAVEL

Douglas Nassif Roma Junior

/douglasjunior

/in/douglasjunior

douglasjunior.me

nassifrroma@gmail.com

Slides: https://git.io/vAd6S

AGENDA

- Introdução
- Configuração.
- Queries nativas
- Capturando eventos de query
- Transação
- Query Builder
- Migrações

INTRODUÇÃO

- Uma das vantagens do Laravel é tornar a interação com a base de dados extremamente simples.

- Seja utilizando queries nativas, query builder ou o Eloquent ORM.

- Atualmente o Laravel suporta nativamente os seguintes bancos de dados:
  - MySQL
  - PostgreSQL
  - SQL Server
  - SQLite

CONFIGURAÇÃO

- Toda a configuração da base de dados fica centralizada no arquivo `config/database.php`.

- Adicionalmente, você pode alterar algumas configurações de sua base de dados através do seu arquivo `.env`

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

QUERIES NATIVAS

- Uma vez que você configurou a conexão com a base de dados, você já pode usar o *facade* `DB`.

- O *facade* `DB` fornece métodos para todos os tipos de query:
  - `select`
  - `update`
  - `insert`
  - `delete`
  - `statement`

QUERIES NATIVAS

- Controller de exemplo de execução de query nativa:

```php
namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\DB;

class TarefaController extends Controller
{

  public function index()
  {
    $tarefas = DB::select('select * from tarefas where ativa = ?', [1]);
    return view('tarefas', ['tarefas' => $tarefas]);
  }
}
```

QUERIES NATIVAS

- View de exemplo:

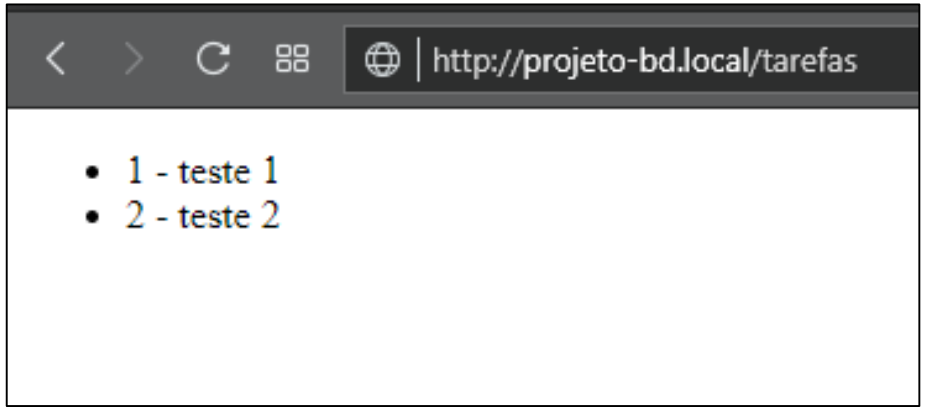
```
<!DOCTYPE html>
<html>
<head>
  ...
</head>
<body>
  <ul>
  @foreach ($tarefas as $tarefa)
    <li>{{ $tarefa->id }} - {{ $tarefa->titulo }}</li>
  @endforeach
  </ul>
</body>
</html>
```

QUERIES NATIVAS

- Rota de exemplo:

```
Route::get('/tarefas', 'TarefaController@index');
```

- Resultado:



QUERIES NATIVAS

- Você também pode trabalhar com parâmetros nomeados:

```php
public function index()
{
  $tarefas = DB::select('select * from tarefas where ativa = :ativa', ['ativa' => 1]);

  return view('tarefas', ['tarefas' => $tarefas]);
}
```

QUERIES NATIVAS

- Além de poder trabalhar com qualquer outro comando SQL:

```
DB::insert('insert into tarefas (titulo, ativa) values (?, ?)', ['Tarefa de teste', 1]);
```

```
$afetados = DB::update('update tarefas set titulo = ?, ativa = ? where id = ?’,
    ['Alterando a tarefa 2', 1, 2]);
```

```
$deletados = DB::delete('delete from tarefas where id = ?', [4]);
```

```
DB::statement('drop table tarefas');
```

CAPTURANDO EVENTOS DE QUERY

- O Laravel também permite que você capture os eventos de todas as queries que forem executadas em sua aplicação.

- Este recurso pode ser interessante para execução de logs ou debug, e para ativar, você deve registrá-lo no **AppServiceProvider**.

```
class AppServiceProvider extends ServiceProvider {
  public function boot() {
    DB::listen(function ($query) {
      // $query->sql
      // $query->bindings
      // $query->time
    });
  }
  ...
}
```

TRANSAÇÕES

- O *facade* `DB` também suporta execução de operações em uma transação.

- Para isso basta utilizar a função `DB::transaction`.

```
DB::transaction(function () {
  DB::delete('delete from tarefas where user_id = ?', [3]);
  DB::delete('delete from usuarios where id = ?', [3]);
});
```

TRANSAÇÕES

- Transações também possuem proteção contra *deadlocks* de banco de dados.

- Para isso você pode passar como segundo argumento da função a quantidade de tempo que a transação deve aguardar caso ocorra um *deadlock* ocorra.
  - Se o tempo estourar, uma exceção será lançada.

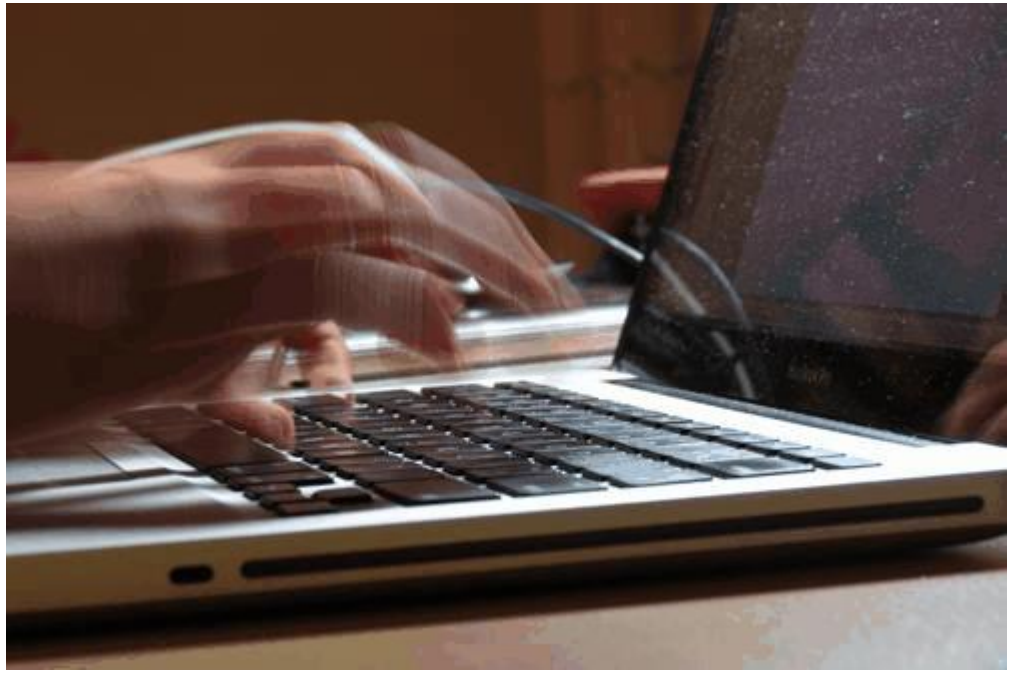
```
DB::transaction(function () {
  DB::delete('delete from tarefas where user_id = ?', [3]);
  DB::delete('delete from usuarios where id = ?', [3]);
}, 5);
```

TRANSAÇÕES

- Você também pode tratar as transações manualmente, para isso basta utilizar as funções **beginTransaction**, **roolBack** e **commit**.

```
try {
  DB::beginTransaction();
  DB::delete('delete from tarefas where user_id = ?', [1]);
  DB::delete('delete from usuarios where id = ?', [1]);
  DB::commit();
} catch (Exception $ex) {
  DB::rollBack();
}
```

VAMOS PRATICAR



QUERY BUILDER

- Laravel **Database Query Builder** fornece uma maneira conveniente e fluente para criar e executar queries no banco de dados.

- Ele pode ser utilizado para executar a maioria das operações de sua aplicação e funciona para todas as base de dados suportadas.

- Adicionalmente, o **Query Builder** utiliza o *binding* de parâmetro do **PDO** para proteger sua aplicação automaticamente de SQL Injections.

QUERY BUILDER

- Para iniciar a construção da query, você deve utilizar a função `table` fornecida pelo *facade* `DB`.

- Por exemplo, para obter todas as tarefas, execute:

```
$tarefas = DB::table('tarefas')->get();
```

QUERY BUILDER

- Se você deseja obter apenas um registro na consulta, você pode utilizar a função `first`.

- Por exemplo:

```
$tarefa = DB::table('tarefas')->where('id', 5)->first();
```

QUERY BUILDER

- Se você quer apenas uma única coluna de um único registro, você pode utilizar a função `value`.

- Por exemplo:

```
$titulo = DB::table('tarefas')->where('id', 5)->value('titulo');
```

QUERY BUILDER

- Se você quer apenas uma única coluna de vários registros, você pode utilizar a função `pluck`.

- Por exemplo:

```
$titulos = DB::table('tarefas')->pluck('titulo');
```

QUERY BUILDER

- Se você deseja trabalhar com consultas que manipulam milhares de registros, você pode precisar da função `chunk`.

- Por exemplo, para consultar de 100 em 100 registros, faça:

```
DB::table('tarefas')->orderBy('id')->chunk(100, function ($tarefas) {
  foreach ($tarefas as $tarefa) {
    //
  }
});
```

QUERY BUILDER

- O **Query Builder** também possui algumas funções de agregação, como `count`, `max`, `min`, `avg` e `sum`.

```php
$quantidade = DB::table('tarefas')->count();

$ultimoId = DB::table('tarefas')->max('id');
```

- E ainda, auxiliares que facilitam checar se determinado registro existe no banco de dados.

```php
return DB::table('tabelas')->where('id', 1)->exists();
```

```php
return DB::table('tabelas')->where('id', 1)->doesntExist();
```

QUERY BUILDER

- Você também pode querer mesclar recursos nativos com o Query Builder, para isso você pode utilizar a função `raw`.

```
$users = DB::table('users’)
  ->select(DB::raw('count(*) as user_count, status’))
  ->where('status', '<>', 1)
  ->groupBy('status’)
  ->get();
```

Atenção: Recursos nativos (raw) serão inseridos na query como *strings*, então você deve ser extremamente cuidadoso para evitar SQL Injetion.

QUERY BUILDER

- Além do `raw`, existem outras funções mais específicas que injetam pedaços de queries nativas durante a construção da query.

- São eles:

```
$orders = DB::table('orders’)
  ->selectRaw('price * ? as price_with_tax', [1.0825])
  ->get();
```

```
$orders = DB::table('orders’)
  ->whereRaw('price > IF(state = "TX", ?, 100)', [200])
  ->get();
```

QUERY BUILDER

- Continuação:

```
$orders = DB::table('orders’)
  ->select('department', DB::raw('SUM(price) as total_sales’))
  ->groupBy('department’)
  ->havingRaw('SUM(price) > 2500’)
  ->get();
```

```
$orders = DB::table('orders')
->orderByRaw('updated_at - created_at DESC')
->get();
```

QUERY BUILDER

- **Query Builder** também dá suporte à criação de junções (joins).

```
$users = DB::table('users’)
  ->join('contacts', 'users.id', '=', 'contacts.user_id’)
  ->join('orders', 'users.id', '=', 'orders.user_id’)
  ->select('users.*', 'contacts.phone', 'orders.price’)
  ->get();
```

```
$users = DB::table('users’)
  ->leftJoin('posts', 'users.id', '=', 'posts.user_id’)
  ->get();
```

```
$users = DB::table('sizes')
->crossJoin('colours')
->get();
```

QUERY BUILDER

- Junções avançadas.

```
DB::table('users’)
  ->join('contacts', function ($join) {
    $join->on('users.id', '=', 'contacts.user_id’)
         ->orOn('users.parent_id', '=', 'contacts.user_id’);
  })
  ->get();
```

```
DB::table('users’)
  ->join('contacts', function ($join) {
    $join->on('users.id', '=', 'contacts.user_id’)
         ->where('contacts.user_id', '>', 5);
  })
  ->get();
```

QUERY BUILDER

- Uniões também estão presentes no **Query Builder**.

```
$first = DB::table('users’)
  ->whereNull('first_name');

$users = DB::table('users’)
  ->whereNull('last_name’)
  ->union($first)
  ->get();
```

QUERY BUILDER

- Uma das cláusulas mais utilizadas no SQL é a `where`, e neste ponto o Query Builder possui uma vasta lista de possibilidades.
  - https://laravel.com/docs/5.6/queries#where-clauses

- Por exemplo:

```
DB::table('users’)
  ->where('name', '=', 'John’)
  ->orWhere(function ($query) {
     $query->where('votes', '>', 100)
           ->where('title', '<>', 'Admin’);
   })
  ->get();
```

```
select * from users where name = 'John' or (votes > 100 and title <> 'Admin')
```

QUERY BUILDER

- E ainda, você também pode aplicar ordenação, agrupamento, `limit` e `offset`.

```php
$users = DB::table('users')->orderBy('name', 'desc')->get();
```

```php
$users = DB::table('users')->groupBy('account_id’)
  ->having('account_id', '>', 100)->get();
```

```php
$users = DB::table('users')->offset(10)
  ->limit(5)->get();
```

QUERY BUILDER

- Não são apenas consultas que podem ser realizadas com o **Query Builder**. Você também pode fazer `inserts`.

```
DB::table('users')->insert(
  ['email' => 'john@example.com', 'votes' => 0]
);
```

```
DB::table('users')->insert([
  ['email' => 'taylor@example.com', 'votes' => 0],
  ['email' => 'dayle@example.com', 'votes' => 0],
]);
```

```
$id = DB::table('users')->insertGetId(
  ['email' => 'john@example.com', 'votes' => 0]
);
```

QUERY BUILDER

- `Updates` e `deletes` também estão presentes:

```php
$qtdAtualizadas = DB::table('tarefas')->where('id', $id)->update([
  'titulo' => $request->titulo,
  'descricao' => $request->descricao,
  'ativa' => $request->ativa,
]);
```

```php
$qtdDeletadas = DB::table('tarefas')->where('id', $id)->delete();
```

MIGRAÇÕES

- Migrações (migartions) são como um controle de versão para sua base de dados.

- Elas permitem que seu time modifique e compartilhe o *schema* da base de dados de sua aplicação.

- Se você já teve que dizer à um colega para adicionar uma nova coluna no banco de dados, então você já enfrentou o problema que as migrações tentam resolver.

MIGRAÇÕES

- **Para criar uma migração, use o comando:**

```
php artisan make:migration criar_tabela_tarefas
```

- **Um arquivo será criado em:**

```
database/migrations/ANO_MES_DIA_HORA_criar_tabela_tar
efas.php
```

MIGRAÇÕES

- Note que nome do arquivo criado contém data e hora, que permite que o Laravel determine a ordem das migrações.

- As opções `--table` e `--create` podem ser usadas para indicar o nome da tabela e que a migração é para a criação de uma nova tabela.

```
php artisan make:migration criar_tabela_tarefas --create=tarefas
```

```
php artisan make:migration adc_descricao_tabela_tarefas --table=tarefas
```

MIGRAÇÕES

- **A classe** `Migration` **contem do is métodos:** `up` **e** `down`

- `up` **é utilizada para realizar a migração, ou seja, adicionar novas tabelas, colunas ou index à sua base de dados.**

- `down` **é utilizado para desfazer a migração, ou seja, destruir tudo que foi criado no** `up`**.**

MIGRAÇÕES

- Nestes métodos, você deve usar o *facade* `Schema` para criar e manipular tabelas de sua base de dados.

```
class CriarTabelaTarefas extends Migration{

  public function up() {
    Schema::create('tarefas', function (Blueprint $table) {
      $table->increments('id’);
      $table->string('titulo', 50);
      $table->tinyInteger('ativa’);
      $table->timestamps();
    });
  }

  public function down() {
    Schema::drop('tarefas’);
  }
}
```

MIGRAÇÕES

- Para rodar as migrações, você deve digitar o seguinte comando:

```
php artisan migrate
```

- E para desfazer uma migração:

```
php artisan migrate:rollback
```

MIGRAÇÕES

- Outras opções úteis para migração são o refresh e o fresh.

- Refresh desfaz todas as migrações e refaz novamente.

```
php artisan migrate:refresh
```

- Fresh remove todas as tabelas e realiza todas as migrações novamente.

```
php artisan migrate:fresh
```

MIGRAÇÕES

- Para criar novas tabelas, você deve usar a função `Schema::create`.

- Esta função aceita dois argumentos, o primeiro é o nome da tabela e o segundo é um função de callback.

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

MIGRAÇÕES

- O facade Schema possui uma série de funções que facilitam o processo de migração, como por exemplo checar se uma tabela ou coluna já existe:

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

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

MIGRAÇÕES

- Você também pode utilizar a migração para definir o tipo de cada coluna.
  - Lista completa de tipos e modificadores: https://laravel.com/docs/5.6/migrations#columns

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

MIGRAÇÕES

- Para modificar colunas, você precisa adicionar um pacote extra chamado doctrine/dbal.

- Esta biblioteca é utilizada para determinar o estado atual da coluna e criar as SQL Queries necessárias para realizar os devidos ajustes na coluna.

```
composer require doctrine/dbal
```

MIGRAÇÕES

- Após incluir a biblioteca, o processo de modificações das colunas continua da forma tradicional.

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

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

Atenção: Somente os seguintes tipos de colunas podem ser alterados: bigInteger, binary, boolean, date, dateTime, dateTimeTz, decimal, integer, json, longText, mediumText, smallInteger, string, text, time, unsignedBigInteger, unsignedInteger and unsignedSmallInteger.

MIGRAÇÕES

- Por fim, as migrações também permitem a manipulação de índices.

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

$table->unique('email');

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

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

- E chaves estrangeiras:

```
Schema::table('posts', function (Blueprint $table) {
  $table->integer('user_id')->unsigned();
  $table->foreign('user_id')->references('id')->on('users');
});
```

DÚVIDAS?

Douglas Nassif Roma Junior

/douglasjunior

/in/douglasjunior

douglasjunior.me

nassifrroma@gmail.com

Slides: https://git.io/vAd6S