02

Migrações e evolução do banco de dados

Nosso sistema já válida de maneira razoável. O que desejemos fazer agora é permitir que outros usuários marquem sua intenção de compra... quero permitir a venda dos produtos do site. Eu, Guilherme, coloco o produto e o Paulo, cliente, pede para comprá-lo, com isso a venda é efetuada.

Precisaremos então de uma tabela no banco representando cada uma dessas vendas: o vendedor, o produto, a data agendada de entrega e o id dela. Poderia entrar no phpmyadmin e executar a query, mas o que aconteceria com meus colegas de trabalho? Eles teriam que executar a mesma query para criar a tabela... como vou notificá-los? E se desejar voltar atrás na minha mudança? E se outro usuário cria outra tabela, como controlar tudo isso?

Para simplificar esse processo de migrar as nossas tabelas já existem bibliotecas, assim como o Code Igniter já possui uma, uma biblioteca de migration. Dentro do nosso projeto, no diretório `application` criamos um chamado `migrations`.

Dentro dele crio um arquivo `001_cria_tabela_de_vendas.php`:

```
<?php
```

Para criar a migração defino sua classe como extensão de uma migração do Code Igniter:

```
<?php
class Migration_Cria_tabela_de_vendas extends CI_migration {
}
```

Agora devemos definir o que acontece quando a migração é executada e para isso definimos a função `up`:

```
<?php
class Migration_Cria_tabela_de_vendas extends CI_migration {
    public function up() {
        }
}
```

Desejamos criar a tabela `vendas`, portanto:

```
    public function up() {
        $this->dbforge->create_table('vendas');
    }
```

Não só isso, desejamos adicionar o campo `id`, do tipo `INT` e auto incrementável:

```
    public function up() {
        $this->dbforge->add_field(array(
            'id' => array(
                'type' => 'INT',
                'auto_increment' => true
```

```
            )
        ));
        $this->dbforge->create_table('vendas');
    }
```

Da mesma maneira adicionamos os campos `produto_id` , `comprador_id` e `data_de_entrega` :

```
    public function up() {
        $this->dbforge->add_field(array(
            'id' => array(
                'type' => 'INT',
                'auto_increment' => true
            ),
            'produto_id' => array (
                'type' => 'INT'
            ),
            'comprador_id' => array(
                'type' => 'INT'
            ),
            'data_de_entrega' => array(
                'type' => 'DATE'
            )
        ));
        $this->dbforge->create_table('vendas');
    }
```

Só falta agora dizer que o campo `id` é nossa chave primária:

```
    public function up() {
        $this->dbforge->add_field(array(
            'id' => array(
                'type' => 'INT',
                'auto_increment' => true
            ),
            'produto_id' => array (
                'type' => 'INT'
            ),
            'comprador_id' => array(
                'type' => 'INT'
            ),
            'data_de_entrega' => array(
                'type' => 'DATE'
            )
        ));
        $this->dbforge->add_key('id', true);
        $this->dbforge->create_table('vendas');
    }
```

E o que fazer se desejamos voltar atrás? Queremos no método `down` remover a tabela `vendas` :

```
    public function down() {
        $this->dbforge->drop_table('vendas');
    }
```

Vamos ativar agora nossas migrações no `migration.php`:

```
$config['migration_enabled'] = TRUE;
```

E dizer para o Code Igniter que o total de migrações que possuímos é 1:

```
$config['migration_version'] = 1;
```

Queremos agora rodar as migrations e para isso vamos criar um controller que as executa, passando por todas elas, invocando as que ainda não foram executadas: criando tabelas, campos, primary keys etc. No diretório `controller` crio o nosso `UtilsController`:

```
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Utils extends CI_Controller {
    public function migrate() {
    }
}
```

A função que devemos invocar se chama `current` na biblioteca `migration`. Então carregamos a biblioteca e chamamos a função:

```
        $this->load->library("migration");
        $success = $this->migration->current();
```

Podemos mostrar um resultado de sucesso ou a mensagem de erro:

```
        if($success) {
            echo 'migrado';
        } else {
            show_error($this->migration->error_string());
        }
```

Acessamos agora `/mercado/utils/migrate` e temos a mensagem de migrado. No nosso banco podemos conferir que a tabela vendas existe com os quatro campos que pedimos.