Ruby On Rails

Engenharia de Software

PUC Minas

Poços de Caldas

Luiz Alberto Ferreira Gomes

Curso de Ciência da Computação

27 de outubro de 2019

Banco de Dados Relacionais (1)

- Um aspecto importante da programação web é a habilidade de coletar, armazenar e recuperar diferentes formas de dados
  - uma das formas mais populares são os bancos de dados relacionais
- Um banco de dados relacional é baseado entidades, denominadas tabelas, no relacionamento, associações, entre elas
- O contêiner fundamental em um banco de dados relacional é denominado de database ou schema
  - podem incluir estruturas de dados, os dados propriamente ditos e permissões de acesso

Banco de Dados Relacionais (2)

- Os dados são armazenados em tabelas e as tabelas são divididas em linhas e colunas. Por exemplo:

Tabela: comment

id	post_id	body
10	1	Ruby realmente...
11	2	Rails facilita...
13	2	Concordo, ...

Banco de Dados Relacionais (3)

- Relacionamentos são estabelecidos entre tabelas para que a consistência dos dados seja mantida em qualquer situação e podem ser:
  - 1:1, 1:N ou N:M

Tabela: comment

id	post_id	body
10	1	Ruby realmente...
11	2	Rails facilita...
13	2	Concordo, ...

Tabela: post

id	title	body
1	A Linguagem Ruby	Ruby é legal.
2	O Framework Rais	O Rais facilita...

I1 - Hora de Colocar as Mãos na Massa (1)

1. Gerar o modelo para os *comentários*

```
$ rails generate model Comment post_id:integer body:text
```

2. Gere a tabela `comment` no banco de dados

```
$ rake db:create
$ rake db:migrate
```

Validação em Aplicações Web

- Validação de Dados é o processo para garantir que a aplicação web operem corretamente. Exemplo:
  - garantir a validação do e-mail, número do telefone e etc
  - garantir que as "regras de negócios"sejam validadas
- A vulnerabilidade mais comum em aplicação web é a injeção SQL

Client Side

- Envolve a verificação de que os formulários HTML sejam preechidos corretamente
  - JavaScript tem sido tradicionalmente utilizado.
  - HTML5 possui "input type"específicos para checagem.
  - Funciona melhor quando combinada com validações do lado do servidor.

Server Side

- A validação é feita após a submissão do formulário HTML
  - banco de dados(stored procedure) - dependente do banco de dados
  - no controlador - veremos mais tarde que não se pode colocar muita lógica no controlador (controladores magros)
  - no modelo - boa maneira de garantir que dados válidos sejam armazenados no banco de dados (database agnostic)
  - Funciona melhor quando combinada com validações do lado do servidor.

Validação em Rails (1)

- Objetos em um sistema OO como tendo um ciclo de vida
  - eles são criaddos, atualizados mais tarde e também destruidos.
- Objetos ActiveRecord têm métodos que podem ser chamados, a fim de assegurar a sua integridade nas várias fases do seu ciclo de vida.
  - garantir que todos os atributos são válidos antes de salvá-lo no banco de dados
- Callbacks são métodos que são invocados em um ponto do ciclo de vida dos objetos ActiveRecord
  - eles são "ganchos"para gatilhos para acionar uma lógica quando houver alterações de seus objetos

Validação em Rails

- Validations são tipo de callbacks que podem ser utilizados para garantir a validade do dado em um banco de dados
- Validação são definidos nos modelos. Exemplo:

```
1  class Person < ApplicationRecord
2      validates_presence_of :name
3      validates_numeracality_of :age, :only_integer => true
4      validates_confirmation_of :email
5      validates_length_of :password, :in => 8..20
6  end
```

I2 - Hora de Colocar a Mão na Massa (1)

- Modifique o arquivo `app/models/post.rb` para exigir que o usuário digite o título e o texto do blog:

```
1          class Post < ApplicationRecord
2            validates_presence_of :title, :body
3          end
```

- Modifique o arquivo `app/models/comment.rb` para exigir que o usuário digite texto do comentário blog:

```
1          class Comment < ApplicationRecord
2           validates_presence_of :body
3          end
```

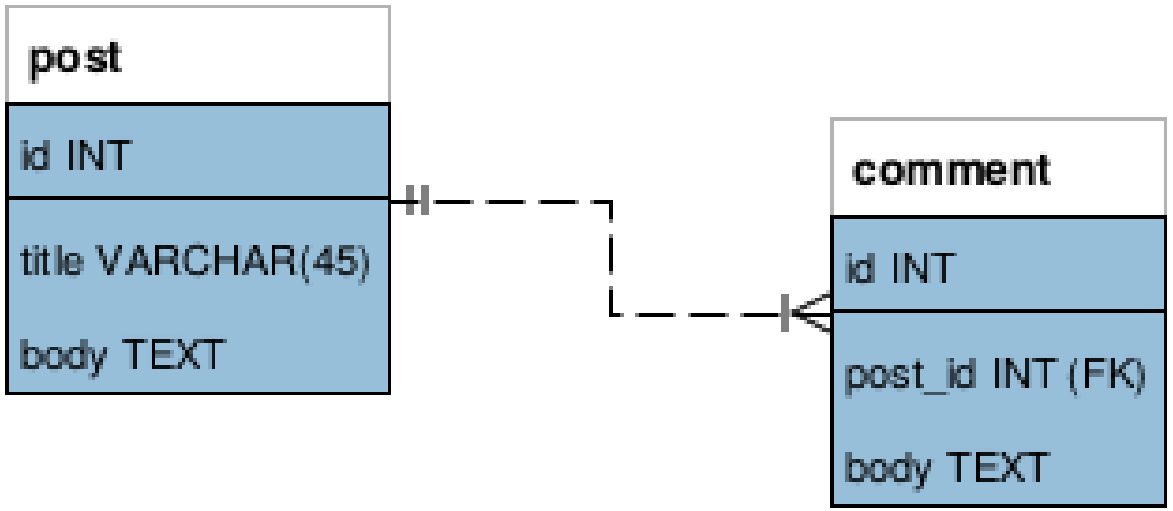
Associações em Rails (1)

- O gerador de modelos utiliza por padrão o ActiveRecord. Isto significa:
  - Tabelas para postagens e comentários foram criadas quando executamos as migrações
  - Um conexão com o banco de dados é estabelecida
  - O ORM é configurado para as postagens e comentátios foi criado - o "M"do MVC.
- No entanto, uma coisa está faltando:
  - tem-se que assegurar que qualquer comentários sejam associados às suas postagens
- Para tornar os modelos em Rails totalmente funcionais precisamos adicionar associações:

Associações em Rails (2)

- cada postagem precisa saber os comentários associado a ele
- cada comentário precisa saber qual é a postagem ele pertence

- Há uma relação muitos-para-um entre comentários e postagens uma:



Associações em Rails (3)

- O ActiveRecord contém um conjunto de métodos de classe para vinculação de objetos por meio de chaves estrangeiras
- Para habilitar isto, deve-se declarar as associações dentro dos modelos usando:

Associação	**Modelo Pai**	**Modelo Filho**
Um-para-um	`has_one`	`belongs_to`
Muitos-para-um	`has_many`	`belongs_to`
Muitos-para-muitos	`has_and_belongs_to_many`	*na tabela junção

I3 - Hora de Colocar a Mão na Massa (1)

- Modifique o arquivo `app/models/post.rb` para associar o post aos seus comentário:

```
1           class Post < ApplicationRecord
2             validates_presence_of :title, :body
3             has_many :comments
4           end
```

- Modifique o arquivo `app/models/comment.rb` para associar o comentário ao seu post:

```
1           class Comment < ApplicationRecord
2             validates_presence_of :body
3             belongs_to :post
4           end
```

Flash (1)

- **Problema:** Queremos redirecionar um usuário para uma página diferente do nosso site, mas ao mesmo tempo fornecer a ele algum tipo de mensagem. Exemplo: "Postagem criada !"
- **Solução:** flash - uma hash onde a dado persiste por exatamente UMA requisição APÓS a requisição corrente
- Um conteúdo pode ser colocado em um flash assim:

Listing 1: controllers/posts_controller.rb

```
1          flash[:attribute] = value
```

- Dois atributos comuns são :notice(good) e :alert (bad)

Flash (2)

- Estes dois atributos (:notice ou :alert) podem ser colocados no redirect_to
- show.html.erb:

Listing 2: views/posts/show.html.erb

```
1           <p id="notice"><%= notice %></p>
2           <p>
3           <strong>Title:</strong>
4           <%= @post.title %>
5           </p>
6           <p>
7           <strong>Body:</strong>
8           <%= @post.body %>
9           </p>
10          <%= link_to 'Edit', edit_post_path(@post) %> |
11          <%= link_to 'Back', posts_path %>
```

Flash (3)

I13 - Hora de Colocar a Mão na Massa (1)

- Modifique o arquivo de rotas para aninhar os comentários às postagens e reinicie o servidor:

Listing 3: config/routes.rb

```
1  Rails.application.routes.draw do
2    resources :posts do
3      resources :comments
4    end
5  end
```

- Modifique o código do template views/posts/show.html.erb. Insira o código abaixo do parágrafo do body.

I13 - Hora de Colocar a Mão na Massa (2)

```
1  <h2>Comments</h2>
2  <div id="comments">
3    <% @post.comments.each do |comment| %>
4     <p>
5    <strong>Posted <%= time_ago_in_words(comment.created_at) %></st
6    <%= h(comment.body) %>
7    </p>
8    <% end %>
9  </div>
```

- Agora no navegador visualize uma postagem que tenha comentários.
- Acrescente o código a seguir logo abaixo do código anterior no arquivo views/posts/show.html.erb:

I13 - Hora de Colocar a Mão na Massa (3)

```
1  <%= form_for([@post, Comment.new]) do |f| %>
2  <p>
3  <%= f.label :body, "New Comment" %><br>
4  <%= f.text_area :body %>
5  </p>
6  <p>
7  <%= f.submit "Add Comments" %>
8  </p>
9  <% end %>
```

- Gere o controlador para os comments:

```
$ rails generate controller comments
```

- Modifique a acão create do controlador controllers/comments_controller.rb:

I13 - Hora de Colocar a Mão na Massa (4)

```
1  before_action :set_comment, only: [:show, :edit, :update, :destro
2
3  def create
4    @post = Post.find(params[:post_id])
5    @comment = @post.comments.create(comment_params)
6
7     if @comment.save
8     redirect_to @post, notice: 'Comment foi criado com sucesso!'
9    else
10     redirect_to @post
11   end
12 end
13
14 private
15    def set_comment
16    @comment = Comment.find(params[:id])
17   end
18
```

I13 - Hora de Colocar a Mão na Massa (5)

```
19    def comment_params
20      params.require(:comment).permit(:body)
21    end
```

- Escolha uma postagem qualquer e escreva alguns comentários.