

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 Rails	O Rails 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: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 preenchidos 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 criados, atualizados mais tarde e também destruídos.
- 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_numericality_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
```

- Inclua as seguintes instruções no início do arquivo `_form.html.erb` para mostrar as mensagens de validação:

I2 - Hora de Colocar a Mão na Massa (2)

Listing 1: app/views/posts/_form.html.erb

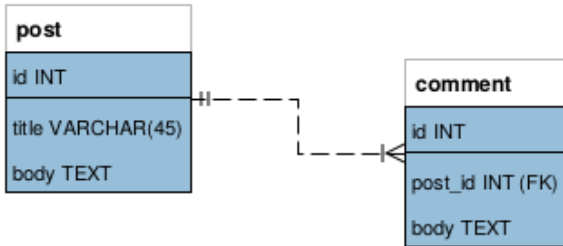
```
1 <% if @post.errors.any? %>
2   <div class="alert alert-danger" role="alert">
3     <h4 class="alert-heading">Request failed!</h4>
4     <ul>
5       <% @post.errors.full_messages.each do |message| %>
6         <li><%= message %></li>
7       <% end %>
8     </ul>
9     <hr>
10    <p class="mb-0">Please, correct issues and try again.</p>
11  </div>
12 <% 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 comentários 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

13 - 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 2: 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`
- Insira após a instrução `<h2>Posts</h2>` `index.html.erb`:

Listing 3: views/posts/index.html.erb

```
1 <%if flash[:notice] %>
2   <div id="notice" class="alert alert-success" role="alert">
3     <%= flash[:notice] %>
4   </div>
5 <% end %>
6 <%if flash[:alert] %>
7   <div id="alert" class="alert alert-danger" role="alert">
8     <%= flash[:alert] %>
9   </div>
10 <% end %>
```

Flash (3)

- Modifique as seguintes ações do controlador `posts_controller.rb` conforme abaixo:

Listing 4: `app/controllers/posts_controller.rb`

```
1  class PostsController < ApplicationController
2    ...
3
4    def create
5      @post = Post.new(post_params)
6      if @post.save
7        redirect_to posts_path,
8          notice: 'Post was sucessfully created!'
9      else
10        render :new
11      end
12    end
```

Flash (4)

```
13
14 def update
15   @post = Post.find(params[:id])
16   if @post.update(post_params)
17     redirect_to posts_path,
18       notice: 'Post was successfully update!'
19   else
20     render :edit
21   end
22 end
23
24 def destroy
25   @post = Post.find(params[:id])
26   if @post.destroy
27     redirect_to posts_path,
28       notice: 'Post was successfully removed!'
29   else
30     redirect_to posts_path,
```

Flash (5)

```
31         alert: 'Fail to remove post!'
32     end
33 end
34 end
```

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

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

Listing 5: config/routes.rb

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

- Modifique o código do template `views/posts/show.html.erb`. Insira o código abaixo do parágrafo da div de fechamento da div panel-body.

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

```
1 <h3>Comments</h3>
2 <div id="comments">
3   <% @post.comments.each do |comment| %>
4     <p>
5       <strong>
6         Posted <%= time_ago_in_words(comment.created_at) %>
7       </strong>
8       <%= h(comment.body) %>
9     </p>
10    <% end %>
11 </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  <div class="card">
2    <%= form_for([@post, Comment.new]) do |f| %>
3      <div class="card-body">
4        <div class="form-group">
5          <%= f.label :body, "New Comment" %><br>
6          <%= f.text_area :body
7            , class: 'form-control', rows: 5 %>
8        </div>
9        <div class="form-group">
10         <%= f.submit "Add Comments"
11           , class: 'btn btn-primary' %>
12        </div>
13      </div>
14    <% end %>
15 </div>
16 <br/>
17 ...
```


l13 - Hora de Colocar a Mão na Massa (4)

- Gere o controlador para os comments:

```
$ rails generate controller comments
```

- Modifique a ação create do controlador `comments_controller.rb`:

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

Listing 6: controllers/comments_controller.rb

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

I13 - Hora de Colocar a Mão na Massa (6)

```
17 private
18   def set_comment
19     @comment = Comment.find(params[:id])
20   end
21
22   def comment_params
23     params.require(:comment).permit(:body)
24   end
25 end
```

- Escolha uma postagem qualquer e escreva alguns comentários.