



Models: ActiveRecord

- ORM (Object-Relational Mapper)
 - Übersetzt Operationen auf Objekten in Datenbankabfragen
- Grundfunktionen
 - Suchen, speichern und löschen in der Datenbank
 - Erzeugung von Gettern und Settern
- Erweiterte Funktionen
 - Assoziationen
 - Validierungen
 - Scopes

. . .

→ Mehr dazu im Verlauf der Veranstaltung!

ActiveRecord: Grundfunktionen

- Rails Console: > rails console
- Erstellen eines Objekts:
 - > Post.create :name => 'Neuer Eintrag'
- Suchen eines Objekts anhand der ID:
 - > Post.find(1)
- Löschen eines Objekts:
 - > post = Post.find(2)
 - > post.destroy

Models: Grundfunktionen

Erzeugen von Gettern und Settern

- > post = Post.find(1)
- > post.name

```
# => "Hello Rails"
```

> post.name = "Hallo Welt"

```
# => "Hallo Welt"
```

> post.save



Erweiterte Funktionen

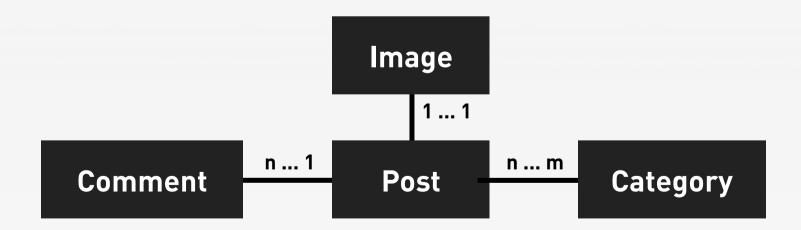
- Assoziationen
- Validierungen
- Callbacks
- Scopes
- "Model-Magic"

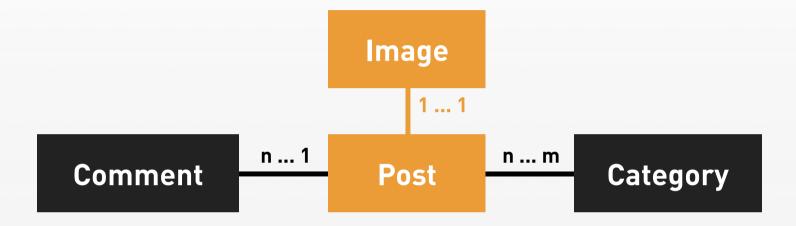


ActiveRecord: Assoziationen

Assoziation:

Eine Assoziation ist eine Verbindung zwischen zwei ActiveRecord-Models. Assoziationen in Rails werden mit Hilfe einer **Domain-specific Language** (DSL) definiert.





> rails generate model image

```
class CreateImages < ActiveRecord::Migration
  def change
    create_table :images do |t|
       t.string :url
       t.timestamps
    end
end</pre>
```

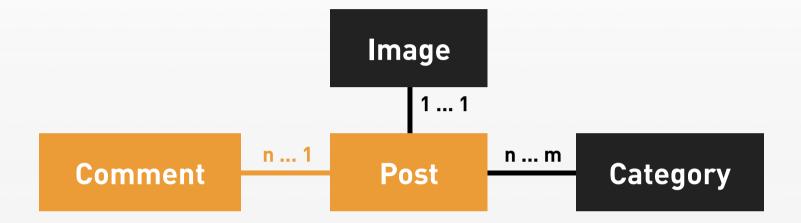
```
class Post < ActiveRecord::Base
has_one :image
end</pre>
```

```
class Image < ActiveRecord::Base
belongs_to :post
end</pre>
```

rails generate migration add_post_id_to_images

```
class AddPostIdToImages < ActiveRecord::Migration
  def change
    add_column :images, :post_id, :integer
  end
end</pre>
```

```
> rails console
> p = Post.find(1)
> p.image
# nil
> p.image = Image.create :url =>
"http:// ..." > p.image.url
```



rails generate model comment body:text

```
class Post < ActiveRecord::Base
has_many :comments
end
```

```
class Comment < ActiveRecord::Base
belongs_to :post
end</pre>
```

rails generate migration add_post_id_to_comments

```
class AddPostIdToComments < ActiveRecord::Migration
  def change
    add_column :comments, :post_id, :integer
  end
end</pre>
```

```
> rails console
> p = Post.find(1)
> p.comments
=> []
> p.comments << Comment.new :body => "Toller Beitrag"
> p.comments
 => [#<Comment id: 1, body: "Toller
Beitrag", created_at: "2014-11-1
22:18:47", updated_at: "2014-11-1
22:18:47", post_id: 1>]
```



- http://railscasts.com/episodes/47-two-many-to-many
- http://quides.rubyonrails.org/association basics.html



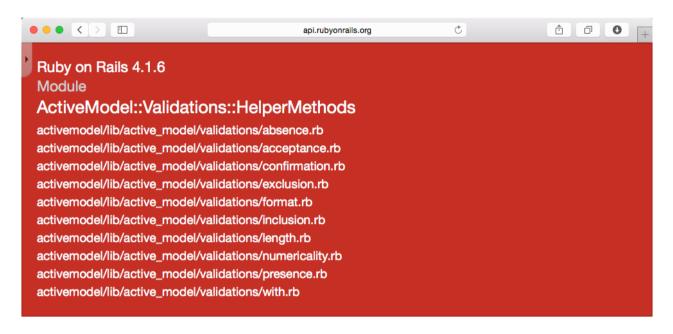
ActiveRecord: Validierungen

Validierungen:

Validierungen stellen sicher, dass nur gültige Objekte in der Datenbank gespeichert werden. Ebenso wie Assoziationen werden Validierungen in Rails mit Hilfe einer **Domain-specific Language** (DSL) definiert.

• Generell:

- Datenbank-Constraints mit Stored Procedures
 Vorteile: "Historisch" bewährt
 Nachteile: Datenbankabhängig, Intransparent, Schwer zu deployen und zu warten
- Client-seitige Validierungen mit z.B. Javascript Vorteile: Unnötige Requests werden vermieden Nachteile: einfach zu umgehen
- Controller-level Validierung
 Nachteile: Schwer zu warten, nicht DRY
- Model-level Validierung
 Vorteile: nicht zu umgehen, DRY, Datenbankagnostisch



Methods

V validates_absence_of, validates_acceptance_of, validates_confirmation_of, validates_exclusion_of, validates_format_of, validates_inclusion_of, validates_length_of, validates_numericality_of, validates_presence_of, validates_size_of

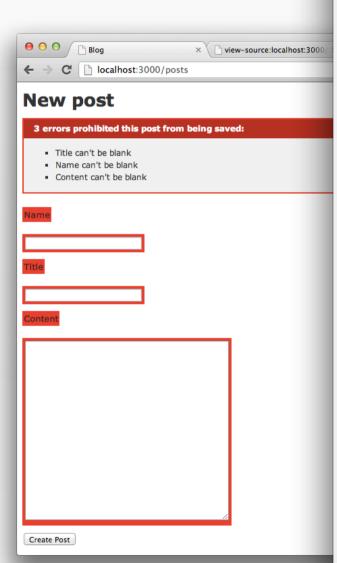
siehe: http://api.rubyonrails.org/classes/ActiveModel/Validations/HelperMethods.html

- Speziell in Rails (Beispiele):
 - Akzeptanz einer Bedingung:
 validates :terms_of_service, :acceptance => true
 - Überprüfung des Eingabeformats (mit Regular Expression): validates_format_of:email,:with => /\A([^@\s]+)@((?:[-a-z0-9]+\.) +[a-z]{2,})\Z/i,:on => :create
 - Überprüfung auf Länge einer Eingabe:
 - validates :name, :length => { :minimum => 2 }
 - validates :bio, :length => { :maximum => 500 }
 - validates :password, :length => { :in => 6..20 }
 - validates :registration_number, :length => { :is => 6 }
 - Prüfung auf Gleichheit von zwei Eingaben:

```
validates :email, :uniqueness => true
```

Speziell in Rails (Beispiele):

```
•••
                          app/models/post.rb
 class Post < ActiveRecord::Base</pre>
   # Prüfung auf "Vorhandensein"
   validates :title, :name, :content, presence: true
 end
```



```
\Theta \Theta \Theta
                                  P view-source:localhost:3000/ ×
          view-source:localhost:3000/posts
     <!IInk nrel= /assets/application.css/pody=1 megla= all rel= stylesneet</pre>
   type="text/css" />
6 6 6 6 7 media="all" rel="stylesheet" type="text/css"
  <link href="/assets/scaffolds.css?body=1" media="all" rel="stylesheet"</pre>
  type="text/css" />
   <script src="/assets/jquery.js?body=1" type="text/javascript"></script>
  <script src="/assets/jquery ujs.js?body=1" type="text/javascript"></script>
  <script src="/assets/posts.js?body=1" type="text/javascript"></script>
  <script src="/assets/application.js?body=1" type="text/javascript"></script>
    <meta content="authenticity token" name="csrf-param" />
  <meta content="+RxHL973aG8jXrWzvAN0RGkv9G7ykDa7NDTdXzavcGY=" name="csrf-token" />
  </head>
  <body>
16
  <h1>New post</h1>
19 <form accept-charset="UTF-8" action="/posts" class="new post" id="new post"
   method="post"><div style="margin:0;padding:0;display:inline"><input name="utf8"
   type="hidden" value="✓" /><input name="authenticity token" type="hidden"
   value="+RxHL973aG8iXrWzvANORGkv9G7vkDa7NDTdXzavcGY=" /></div>
       <div id="error explanation">
20
21
         <h2>3 errors prohibited this post from being saved:</h2>
22
23
24
          Title can't be blank
25
          Name can&#x27:t be blank
26
          Content can't be blank
        </111>
27
28
      </div>
29
    <div class="field">
30
       <div class="field with errors"><label for="post name">Name</label></div><br/>>
31
       <div class="field with errors"><input id="post name" name="post[name]" size="30"</pre>
32
   type="text" value="" /></div>
33
    <div class="field">
34
       <div class="field with errors"><label for="post title">Title</label></div><br />
       <div class="field with errors"><input id="post title" name="post[title]"</pre>
  size="30" type="text" value="" /></div>
    </div>
37
    <div class="field">
       <div class="field with errors"><label
   for="post content">Content</label></div><br />
       <div class="field_with_errors"><textarea cols="40" id="post content"</pre>
   name="post[content]" rows="20">
  </textarea></div>
```

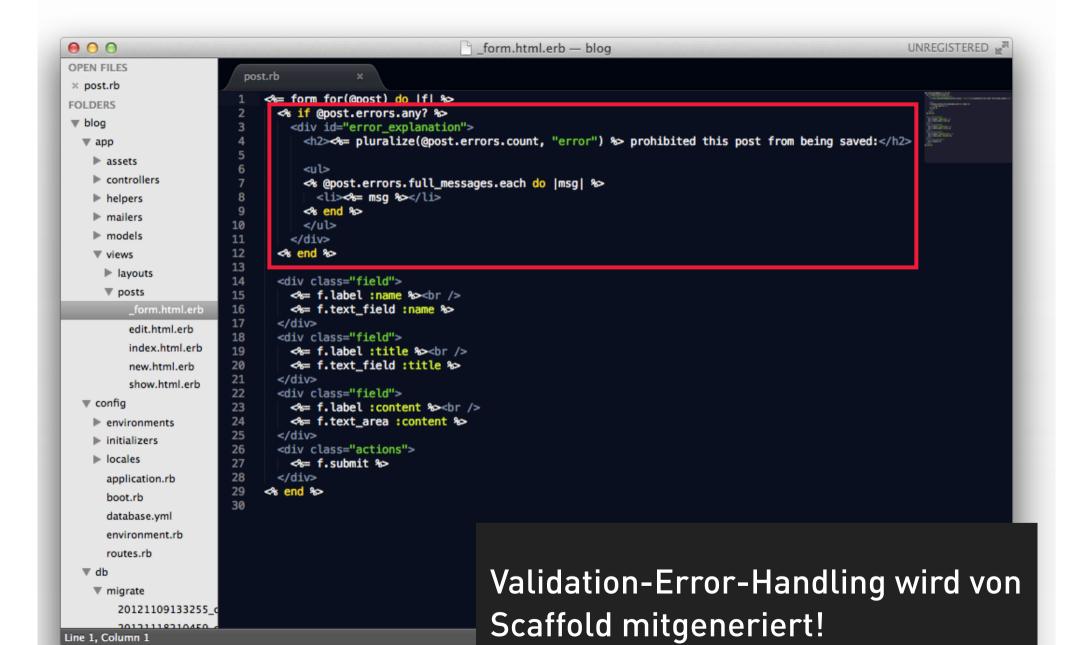
Funktionsweise von Validierungen

 Validierungen werden nach folgenden Methoden aufgerufen:

```
create, create!, save, save!, update,
update_attributes, update_attributes!
```

- Validierungsfehler k\u00f6nnen auf Model-Ebene abgefragt werden:
 - > Post.new.errors.any? # => false
 - > Post.create.errors.any? # => true

siehe: http://guides.rubyonrails.org/active_record_validations_callbacks.html





Callbacks in Rails

Callbacks:

Methoden, die in bestimmten Situationen des Objekt-Lebenszyklus automatisch aufgerufen werden.

Beispiele:

- Nach Anlegen eines Objekts:
 - before_validation, after_validation, before_save, around_save, before_create, around_create, after_create, after_save
- Nach Aktualisierung eines Objekts:
 - before_validation, after_validation, before_save, around_save, before_update, around_update, after_update, after_save
- Nach Löschen eines Objekts:
 before_destroy, around_destroy, after_destroy

Funktionsweise von Callbacks

rails generate migration add_comment_counter_column

```
db/migrate/20121119101134 add comment counter column.rb
class AddCommentCounterColumn < ActiveRecord::Migration</pre>
  def up
    add_column :posts, :comment_count, :integer, :default => 0
  end
  def down
   remove_column :posts, :comment_count
  end
end
```

Funktionsweise von Callbacks

```
•••
                           app/models/comment.rb
 class Comment < ActiveRecord::Base</pre>
   attr_accessible :body, :post_id
   belongs_to :post
   after_create :increment_counter
   def increment counter
     self.post.update_attribute(:comment_count,
 self.post.comments.size)
   end
 end
```

Funktionsweise von Callbacks

- > p = Post.first
- > Comment.create(:body => "Ein Kommentar", :post_id => p.id)

siehe: http://guides.rubyonrails.org/active_record_validations_callbacks.html



ActiveRecord: Scopes

- Conditions
- Joining
- Custom Scopes

siehe: http://guides.rubyonrails.org/active_record_querying.html

Conditions

- Einfache Abfragen:
 - > Post.first
 - > Post.last
 - Post.all
 - > Post.find_all_by_name("Neuer Eintrag")

Conditions

- Abfragemethoden: where, select, group, order, reorder, reverse_order, limit, offset, joins, includes, lock, readonly, from, having
- > Post.where("name like ?", "%eu%")
- > Post.select("id").all

Joining

- Post.comments
- Post.joins(:comments)

Custom Scopes

Custom Scope:

Abfrage, die sich über eine selbst definierte Methode aufrufen lässt.

```
app/models/post.rb

class Post < ActiveRecord::Base
  scope :published, -> { where(:published => true) }
end
```

- Erst Spalte "published" per Migration hinzufügen, dann:
 - > Post.published



Counter Cache

```
•••
                          app/models/comment.rb
 class Comment < ActiveRecord::Base</pre>
     attr_accessible :body, :post_id
     # Counter Cache aktualisiert automatisch den Counter
     belongs_to :post, :counter_cache => :comment_count
     # after_create :increment_counter
     # def increment counter
       self.post.update_attribute(:comment_count,
   self.post.comments.size)
     # end
 end
```



ActionController

```
\Theta \Theta \Theta
                                                               posts controller.rb — blog
                                                                                                                                          UNREGISTERED W
OPEN FILES
                            post.rb × 20121119101134 add ( × 20121118221232_add | × comment.rb × 20121119111313_add | ×
                                                                                                                                     posts_controller.rb ×
× post.rb
                                class PostsController < ApplicationController</pre>
× 20121119101134 add cor
                                  # GET /posts
× 20121118221232 add po
                                  # GET /posts.json
× comment.rb
                                     @posts = Post.all
× 20121119111313 add pu
× posts_controller.rb
                                     respond_to do |format|
                                       format.html # index.html.erb
FOLDERS
                                       format.json { render json: @posts }
10

    app

                                  end
    assets
                                  # GET /posts/1

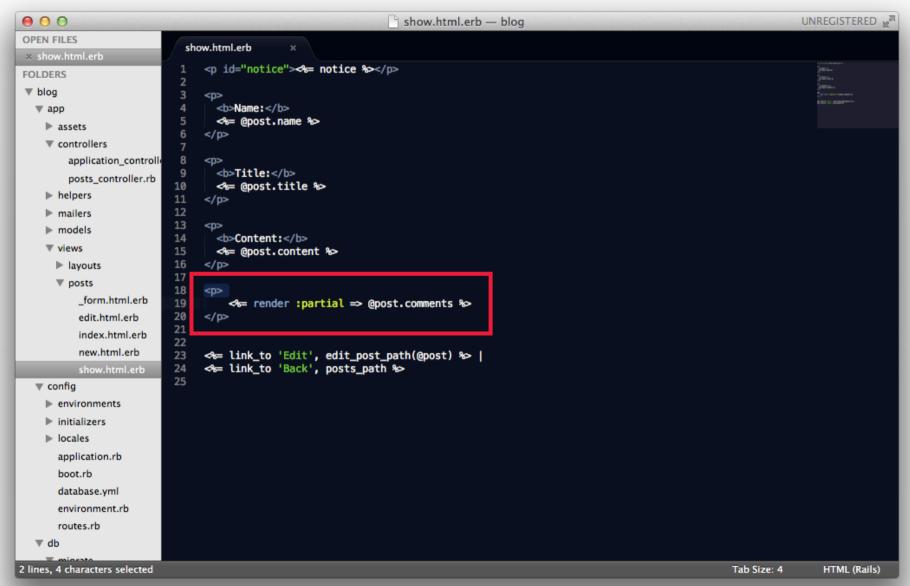
▼ controllers

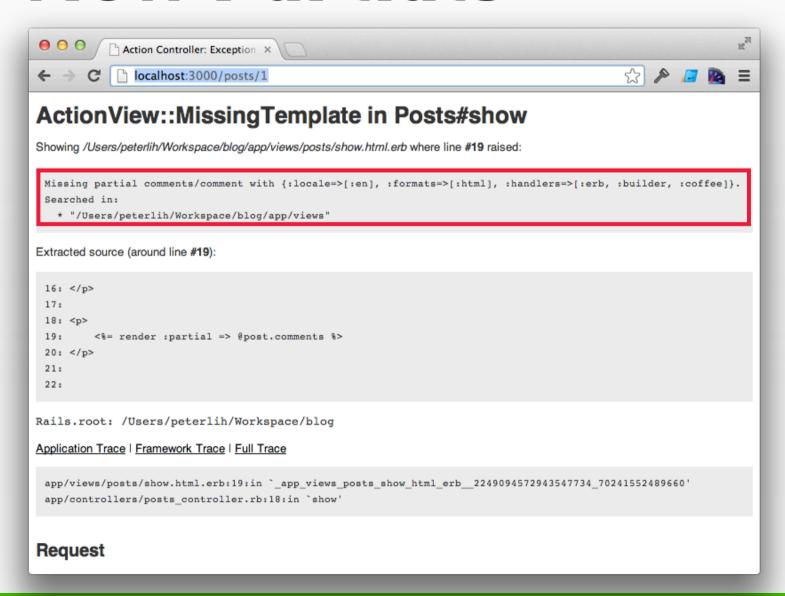
                                  # GET /posts/1.json
        application controll
                                     @post = Post.find(params[:id])
        posts_controller.rb
    helpers
                                     respond_to do |format|
    ▶ mailers
                           19
                                       format.html # show.html.erb
                           20
                                       format.json { render json: @post }
    ▶ models
    end
      ▶ layouts
                           24
                                  # GET /posts/new

▼ posts

                                  # GET /posts/new.json
          form.html.erb
                           26
                                  def new
          edit.html.erb
                                    @post = Post.new
          index.html.erb
                           28
                           29
                                     respond_to do |format|
          new.html.erb
                           30
                                       format.html # new.html.erb
          show.html.erb
                                       format.json { render json: @post }

▼ config
                                  end
    environments
                           34
    initializers
                                  # GET /posts/1/edit
                           36
                                  def edit
    locales
                                    @post = Post.find(params[:id])
      application.rb
                           38
Line 1, Column 1
                                                                                                                              Spaces: 2
                                                                                                                                                 Ruby
```





- > mkdir app/views/comments
- > touch app/views/comments/_comment.html.erb

```
app/views/comments/_comment.html.erb

<%= comment.body %><br/>
```

- Durch Übergabe einer Collection an das Partial, automatische Iteration über das Partial so lange noch weitere Einträge in der Collection sind!
- Varianten: (...)

