Ruby on Rails
Active Record

MySQL-UG
Hamburg - 06.11.2006

Stefan Saasen <s@juretta.com>

Rails. Überblick.

- Framework zur Entwicklung datenbankbasierter Webapplikationen.
- Geschrieben in Ruby.
- Implementiert das Model-View-Controller-Pattern.
- Grundlegende Prinzipien: „Don‘t repeat yourself (DRY)“, „Convention over configuration“
- Codegenerierung, Bereitstellung von Infrastruktur (Logging, Debugging), Pluginarchitektur, Integriertes Testframework!

Ruby.

- Dynamische, sehr kompakte Skriptsprache.
- Vollständig objektorientiert - auch primitive Datentypen, selbst Klassen sind Objekte.
- Folgt der Syntax menschlicher Sprache: Car.new.start_engine unless Road.closed?
- „Principle of Least Surprise“

Ruby. Syntax.

```ruby
class Person
  # Erzeugt getter und setter für @name und @age
  attr_accessor :age, :name
  attr_reader :created
  @@instances = 0
  # Konstruktor
  def initialize(name, age)
      @name, @age = name, age # Parallele Zuweisung
      @created = Time.now
  end
  # Kommentar
  def age_in_days
      # Rückgabewert ist automatisch die letzte Auswertung
      @age * 360 # naive Implementierung
  end
end
class Customer < Person; end
# Klammern können weggelassen werden
p = Person.new("Donald", 60) # p = Customer.new "Kunde", 29
puts p.age => 60
puts p.created => Sun Nov 05 16:09:14 CET 2006
```

Ruby. Blöcke und Iteratoren.

```
(1..20).reject {|zahl| zahl.modulo(2) == 0}
=> [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

%w(Auto Katze Hund Autobus).each do |etwas|
  puts "#{etwas} ist ein Tier" if etwas !~ /Auto/
end
=> Katze ist ein Tier
Hund ist ein Tier
```

```
1.step(100, 10) do |i|
  print "#{i} "
end
=> 1 11 21 31 41 51 61 71 81 91

File.open("/Users/stefan/.bash_profile") do |f|
  print f.read
end
```

Reflection. Ruby ist offen.

```ruby
# Welche Nachrichten (Methoden) versteht ein Objekt?
  "hallo welt".methods => ["%", "index", "select", ...

# Versteht ein Objekt eine bestimmte Nachricht
  [1,2].respond_to?(:<<) => true

# Klassen (auch Kernklassen) können geöffnet/verändert
# werden
class Numeric
  def megabytes; self * 1024 * 1024; end
end
2.megabytes => 2097152

class String
  def words; self.split(" ").size; end
end
puts "Das sind wohl fünf Wörter".words => 5
```

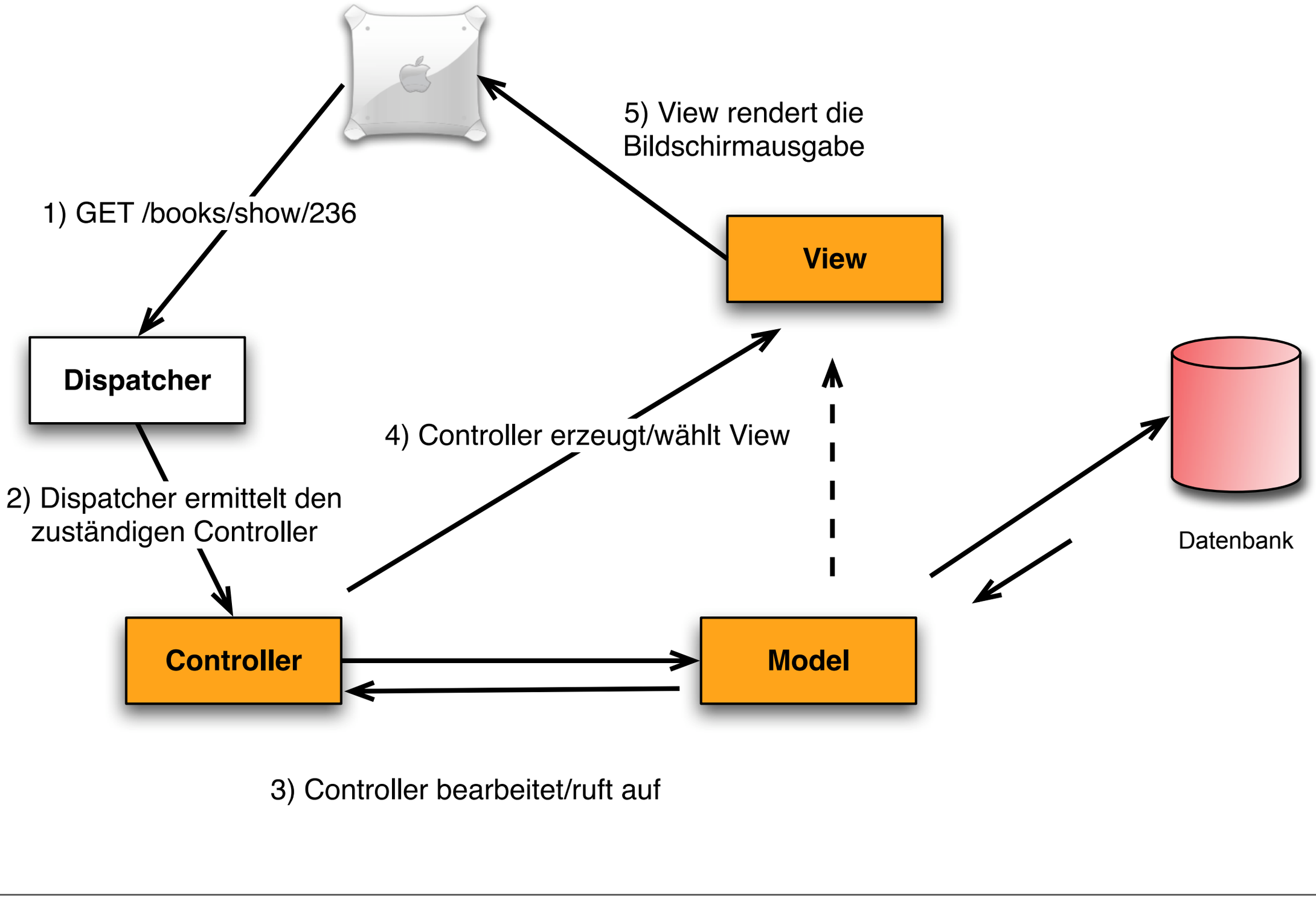
Rails. Überblick.

- Framework zur Entwicklung datenbankbasierter Webapplikationen.
- Geschrieben in Ruby.
- Implementiert das **Model-View-Controller-Pattern**.
- Grundlegende Prinzipien: „Don‘t repeat yourself (DRY)“, „Convention over configuration“
- Codegenerierung, Bereitstellung von Infrastruktur (Logging, Debugging), Pluginarchitektur, Integriertes Testframework!

Model-View-Controller.

- Architekturmuster zur Trennung eines Programms in die drei Einheiten Datenmodell (Model), Präsentation (View) und Programmsteuerung (Controller).
- Model: Objekte/Klassen der Betrachtungsdomäne - "Geschäftsklassen"
- View: (R)HTML-Templates
- Controller: Verarbeitet Anfragen des Clients, Verwaltet die Modelle und zeigt Modelleigenschaften in den Views an.

Rails. MVC.


1) GET /books/show/236
Dispatcher
2) Dispatcher ermittelt den zuständigen Controller
Controller
3) Controller bearbeitet/ruft auf
Model
4) Controller erzeugt/wählt View
View
5) View rendert die Bildschirmausgabe
Datenbank

Rails. Philospohie.

- „Don‘t repeat yourself“
- „Convention over configuration“
  Konfigurationen, wie die Zuordnung von Modellen zu Datenbanktabellen, die andere Webframeworks oft mittels komplexer XML-Dateien herstellen, werden in Ruby on Rails über Konventionen abgebildet.

Rails. Bestandteile.

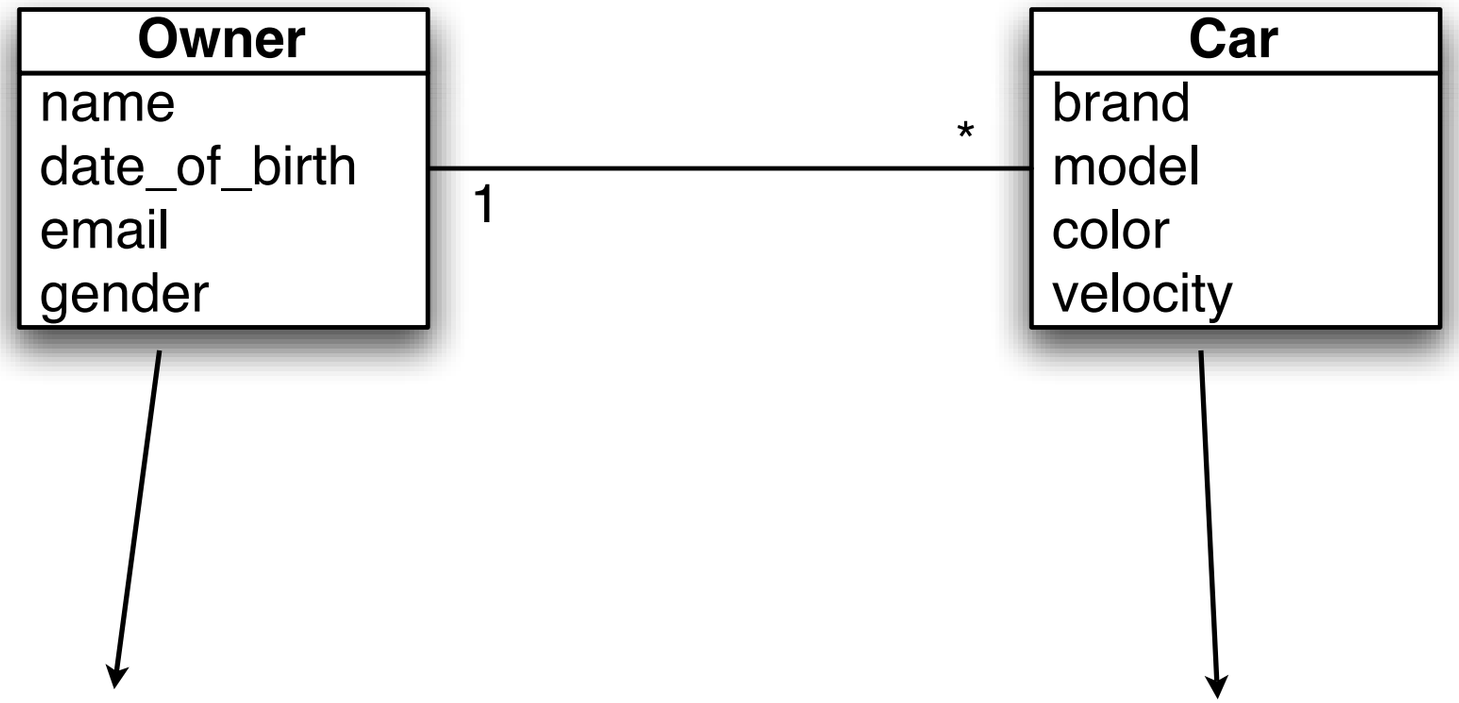
- **Active Record**: Objektrelationales Mapping und Datenbankabstraktion
- **Action Pack**: Controller und Views
- **Active Support**: Erweiterung von Ruby
- **Action Mailer**: Versand und Empfang von E-Mails innerhalb der Anwendung
- **Action Web Service**: Webservices auf Basis von SOAP und XML-RPC

Rails. Active Record.

- Objektrelationales Mapping. Mapping der Klassen und -attribute auf Datenbanktabellen. Mapping der Assoziationen auf Fremdschlüsselbeziehungen.
- Kapselt Datenbankzugriffe.
- Stellt eine Zeile in einer Datenbanktabelle dar.
- Verwaltet Beziehungen zu assoziierten Objekten.
- Domänenspezifische Verarbeitungslogik im Model.

Beispielklassen.

Klassen



Tabellen

```
CREATE TABLE `owners` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(100) NOT NULL,
  `date_of_birth` datetime NOT NULL,
  `email` varchar(100) NOT NULL,
  `gender` varchar(1) NOT NULL default 'm',
  PRIMARY KEY  (`id`)
)
```

```
CREATE TABLE `cars` (
  `id` int(11) NOT NULL auto_increment,
  `owner_id` int(11) NOT NULL,
  `brand` varchar(100) NOT NULL,
  `model` varchar(100) NOT NULL,
  `color` varchar(100) NOT NULL,
  PRIMARY KEY  (`id`),
  KEY `cars_owner_id_index` (`owner_id`)
)
```

Active-Record (AR).

```
# Owner
class Owner < ActiveRecord::Base; end
# Car
class Car < ActiveRecord::Base; end

# Create: Erzeugen und Speichern des Objektes
owner = Owner.create :name => "Dagobert Duck",
      :email => "dagobert@entenhausen.com",
      :gender => 'm', :date_of_birth => 30.years.ago
puts owner.inspect
=> #<Owner:0x61facc @attributes={"name"=>"Dagobert
Duck", "gender"=>"m", "id"=>"17",
"date_of_birth"=>"1976-11-04 05:26:28",
"email"=>"dagobert@entenhausen.com"}>
owner.name = "Donald Duck"
owner.save
```

Active-Record (AR).

```
# Owner
class Owner < ActiveRecord::Base; end
# Car
class Car < ActiveRecord::Base; end
# Finder
Owner.find(:all) => [<Owner...>] # Array von Owner-
Objekten
```

```
owner = Owner.find(12)
owner = Owner.find(:first, :conditions =>
["created_at < ?", 30.days.ago]

# Dynamische Finder
owner = Owner.find_by_name_and_email "Micky Mouse",
"nomail@example.com"
owner.nil? => true
```

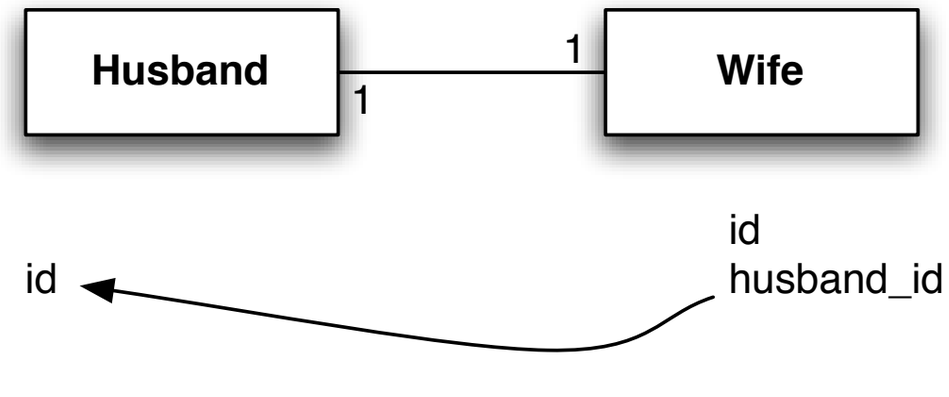
Active-Record (AR).

- Methoden auf Instanzebene beziehen sich auf konkrete Datensätze (Zeilen)
- Klassenmethoden beziehen sich auf die gesamten Daten (Tabelle)
  - Owner.count
  - Owner.average('date_of_birth')
  - Owner.maximum('date_of_birth')
  - Owner.delete_all oder Owner.destroy_all

AR. Assoziationen.

- Assoziationen werden in Datenbanken über Fremdschlüsselbeziehungen zwischen Tabellen aufgelöst.
- Drei Grundformen der Assoziationen:
  - 1 zu 1 (Husband <-> Wife)
  - 1 zu n (Owner <-> Car)
  - n zu n (Author <-> Book)

AR. Assoziationen.



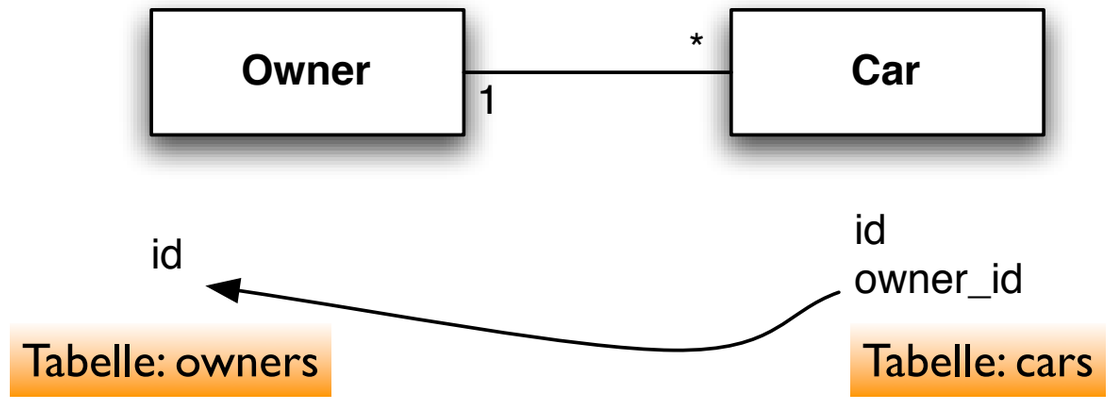


Tabelle: owners

Tabelle: cars

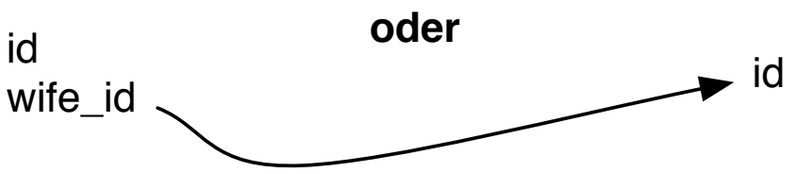


Tabelle: husbands

Tabelle: wifes

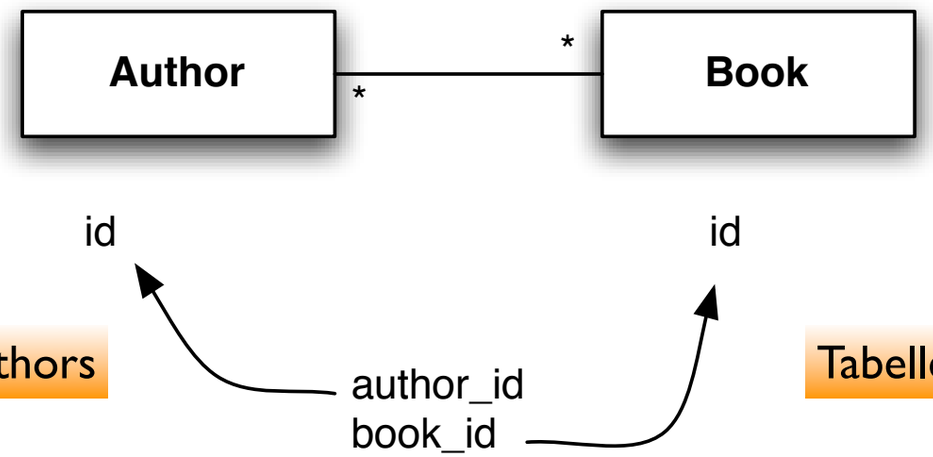


Tabelle: authors

Tabelle: books

Tabelle: authors_books

AR. Assoziationen. f.

- Abbildungen von Assoziationen über den Aufruf von Klassenmethoden (Makros).
- has_one
- belongs_to
- has_many
- has_and_belongs_to_many

Active-Record (AR).

```ruby
# Owner
class Owner < ActiveRecord::Base
  # Assoziierte Objekte werden gelöscht wenn das
  # zugehörige Owner-Objekt gelöscht wird.
  has_many :cars, :dependent => :destroy
end

# Car: belongs_to -> Fremdschlüssel owner_id
class Car < ActiveRecord::Base;
  belongs_to :owner
end

# Owner hat nun zusätzliche Methoden z.B.:
# cars=, cars, cars<<, cars.empty?...
Owner.find(17).cars.create(:model => "S-Klasse", :brand =>
"Merceds", :color => "black", :velocity => 250)
Owner.find(17).cars => [...]
# Car hat analog Methoden wie owner, owner=, owner?
```

Validierung.

- Validerung von Daten im Modell **nicht** im View oder im Controller!
- Active Record bietet hierfür zahlreiche Validierungsmethoden:
  - Pflichtfelder: validates_presence_of
  - Sind assoziierte Objekte in sich gültig: validates_associated
  - Doppeleingaben (Passwort bspw.) überprüfen: validates_confirmation_of
  - Matcht ein regulärer Ausdruck?: validates_format_of
  - Länge der Eingabe überprüfen: validates_length_of
  - ...

Active-Record (AR).

```ruby
# Owner
class Owner < ActiveRecord::Base
  has_many :cars
  # Name und E-Mail sind Pflichtfelder
  validates_presence_of :name
  # E-Mail muss das Format einer E-Mail haben
  validates_format_of :email, :with => /^([^@\s]+)@((?:
  [-a-z0-9]+\.)+[a-z]{2,})$/i, :on => :create
end
owner = Owner.new
owner.save # => false
owner.errors.empty?                  # => false
owner.errors.count                   # => 2
owner.errors.on "name"               # => "can't be empty"
owner.errors.each_full { |msg| print msg }
# => Name can't be blank Email is invalid
```

Active-Record (AR).

```ruby
class Car < ActiveRecord::Base
  belongs_to :owner
  validates_length_of :brand, :within => 3..30
  validates_exclusion_of :color, :in => %w(white grey)
  validates_numericality_of :velocity
  validate_presence_of :owner_id
end
```

```ruby
car = Car.new :brand => "Audi", :color => "pink"
car.velocity = "100 km/h"
car.save # => false
car.velocity = 100
car.owner = Owner.find(:first)
car.save # => true
```

AR.Vererbung.

- Active Record unterstützt ausschließlich "Single-Table-Inheritance".
- Alle Attribute der Vererbungshierachie werden in einer Tabelle gespeichert. Das zusätzliche Attribut "type" speichert den abzuspeichernden Typ.

AR. Vererbung.

```
class Car < ActiveRecord::Base
  # [...]
end

class Racecar < Car
  def accelerate!
    velocity * 1.6
  end
end

ferrari = Racecar.find(34)
ferrari.accelerate!
```

AR. Callbacks.

- Callbacks ermöglichen die Ausführung von Methoden innerhalb des Lebenszyklus der Objekte.

- Beispiele für Callbacks:
  - before_validation
  - after_save
- Callbacks auch über externe Observer möglich.

AR. Callbacks.

```
require 'md5'
class Car < ActiveRecord::Base
  before_create :create_identifier

  private
    def create_identifier
      write_attribute("identifier", MD5.hexdigest(rand
  (1000).to_s + Time.now.to_s))
    end
end # class Car
# Externer Observer
class CarObserver < ActiveRecord::Observer
  def after_destroy(car)
    Notification.deliver_admin_email("Car destroyed")
  end
end
```

AR. Transaktionen.

- Active Record unterstützt Transaktionen sowohl auf Objekt-, als auch auf Datenbankebene.

AR. Transaktionen.

```
dagobert  = Owner.find_by_name "Dagobert Duck"
daisy     = Owner.find_by_name "Daisy Duck"

Owner.transaction(dagobert, daisy) do
  cars = daisy.cars
  dagobert.cars << cars
  dagobert.save
  daisy.cars.clear
end

# Operationen werden innerhalb einer Transaktion
# ausgeführt.
```

AR.Acts_as.

- acts_as_list: Unterstützt die Sortierung von Objekten. Die zugehörige Tabelle enthält ein Feld "position". Fügt Methoden wie "move_to_bottom" oder "move_higher" hinzu.
- acts_as_tree: Bildet Baumstrukturen ab über Eltern-Kind-Beziehungen ab.
- acts_as_nested_set: Ähnlich der Baumstruktur des acts_as_tree-Mixins. Kinder und Enkelkinder eines Knotens können aber hier mit einer einzigen Abfrage ausgelesen werden.
- Plugins erweitern die Funktionalität der AR-Klassen. Bsp.: acts_as_taggable, acts_as_versioned...

AR. Mehr.

- Serialisierung von Datenstrukturen über "serialize". Sinnvoll bei Arrays oder Hashes.
- Timestamps: Datenbankfelder die "created_at", "created_on", "updated_at" oder "updated_on" heißen, werden automatisch belegt.
- Aggregationen möglich. Attribute einer AR-Klasse können als Value-Objekte abgebildet werden.
- Observer können den Lebenszyklus von Objekten beobachten und auf Ereignisse reagieren.

AR. Mehr. f.

- Caching von assoziierten Objekten.
- Eager Loading. Verhindert das Auftreten von 1+N-Problemen.
  Owner.find(:all, :include => :cars)
- AR unterstützt optimistic-locking wenn das Feld "lock_version" in der Tabelle vorhanden ist.

Unit-Tests.

- Rails unterstützt integriertes Testen.
- Bei der Erzeugung von Controller oder Model-Klassen werden automatisch Test-Cases angelegt.
- Testen von Controllern, Model und Views.
- Integration-Tests erlauben das Testen von vollständigen Prozessen.

Migrations.

- Data Definition Language in Ruby
- Aktualisieren des Datenbankschemas und ggf. von Testdaten.
- Die Veränderung des Datenbankschemas ist versionierbar.
- Unterstützt verteiltes Entwickeln.

Rails in 10 Minuten.

- Projekt erzeugen
- Datenbank anlegen
- Verbindungsparameter definieren. Genug der Konfiguration.
- Server starten

Rails. Das haben wir nicht gesehen.

- Integriertes Caching in Views/Controllern.
- Pluginarchitektur.
- Debugging.
- Unterstützung von Ajax in den Views
- Helper

Ende

- Fragen?