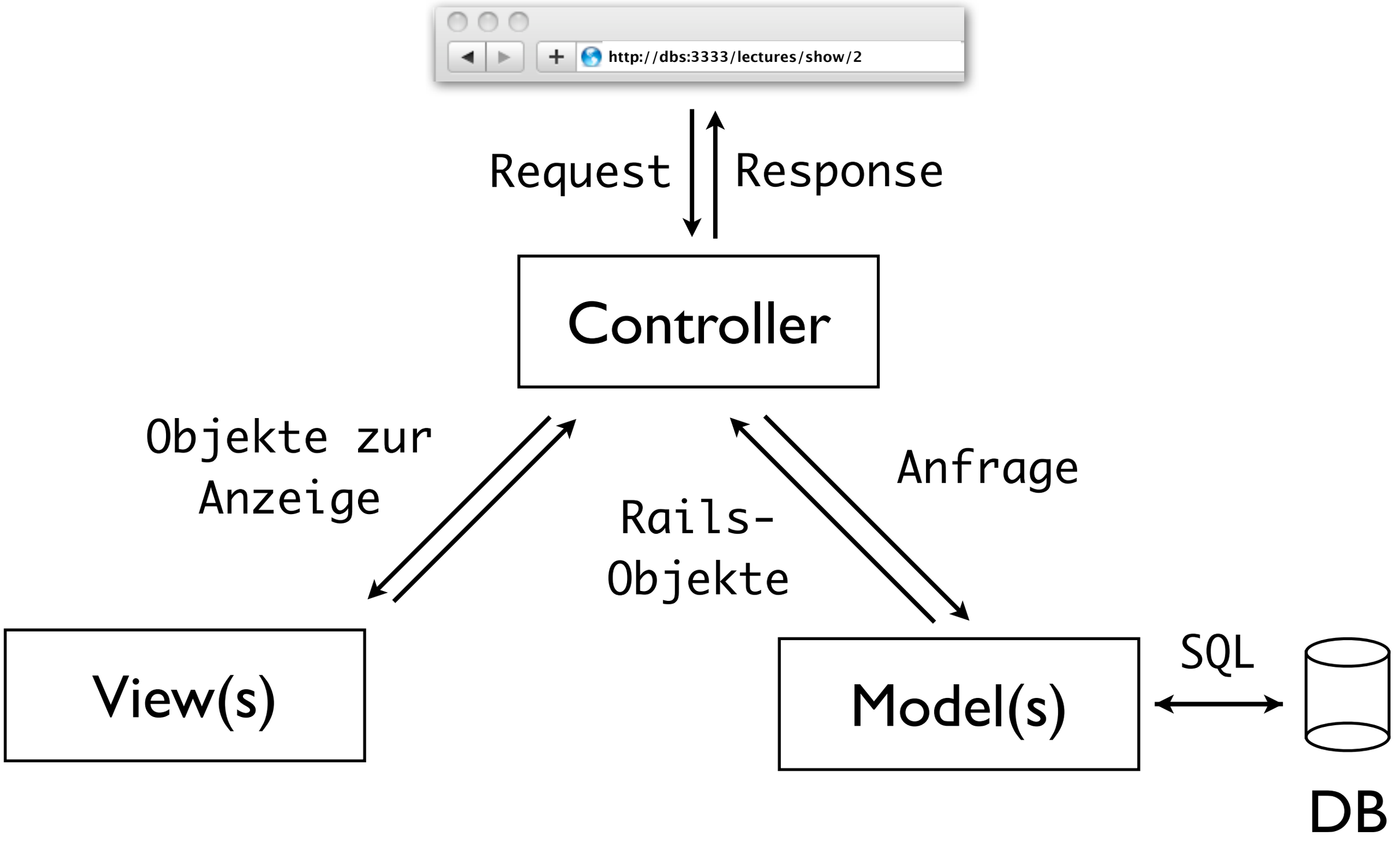
Datenbankapplikationen mit Ruby on Rails, Teil 2

Datenbanksysteme 2011
Universität Osnabrück
Gastvorlesung von Nicolas Neubauer

Views & Templates



Wiederholung

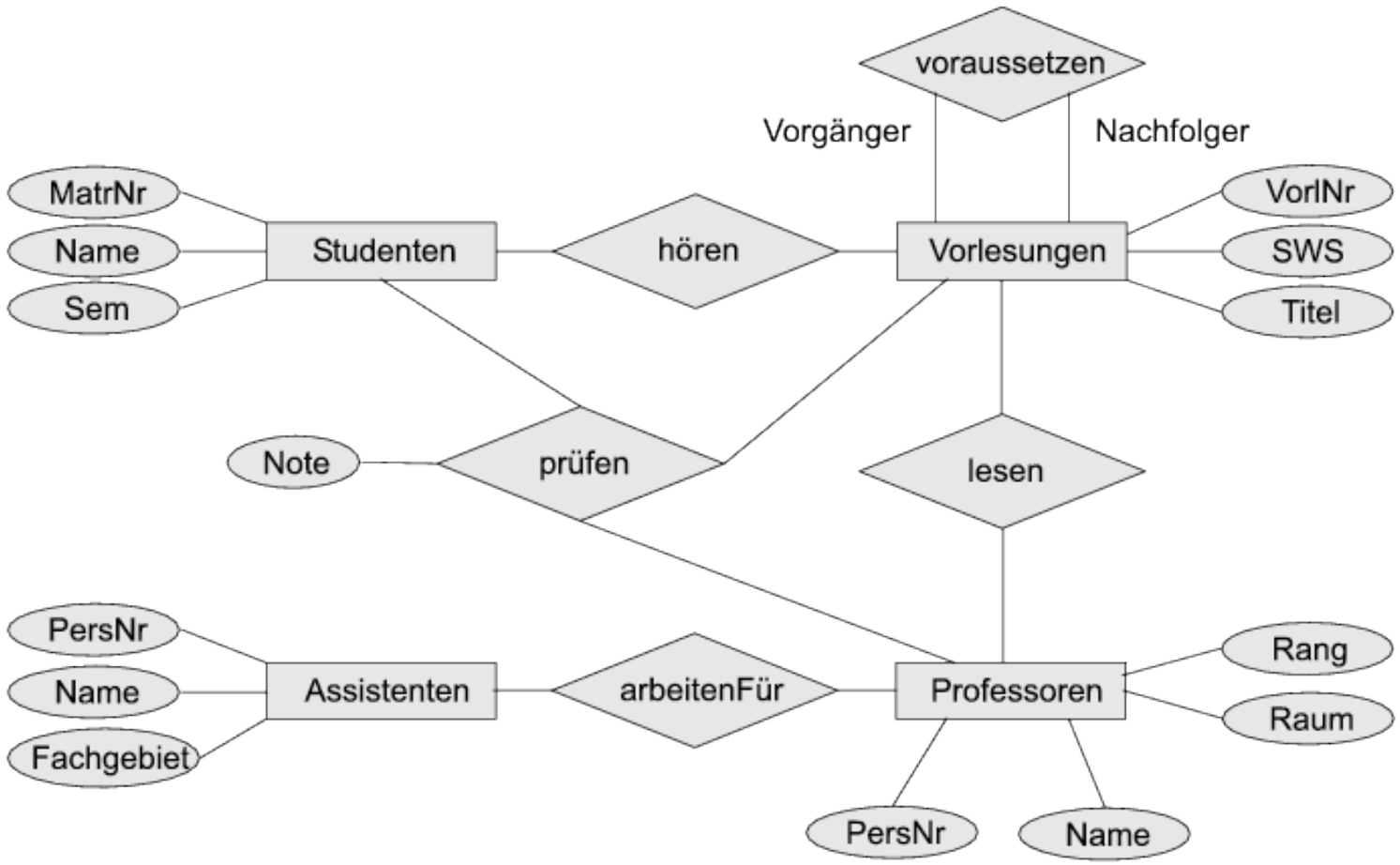
Demo

Wiederholung

- Model-View-Controller
- Don‘t Repeat Yourself
- Convention over Configuration
- Objektrelationales Mapping
- Browser-Request Verarbeitung
- Templates

Neue Applikation von 0

- Bisher: Vorhandene Datenbank
- Jetzt: Komplett neue Applikation



Scaffolds

- Scaffold = Gerüst
- `rails generate scaffold lectures`
- erzeugt:
  - Migration
  - Model
  - Controller
  - Views
  - Tests

Migration

- Methode für iterative Datenbankentwicklung

```
class CreateLectures < ActiveRecord::Migration
  def self.up
    create_table :lectures do |t|
      t.string  :titel
      i.integer :vorl_nr
    end
  end

  def self.down
    drop_table :lectures
  end
end
```

Migration

- Methode für iterative Datenbankentwicklung

```
class AddSWSLectures < ActiveRecord::Migration
  def self.up
    add_column :lectures, :sws, :integer
  end

  def self.down
    remove_column :lectures, :sws
  end
end
```

Migration

- übernimmt später CREATE/ALTER TABLE

```
$ rails generate scaffold lectures
  titel:string sws:integer vorl_nr:integer
  professor:references
```

```
DATENFELD:DATENTYP
```

```
string
text
integer
float
datetime
...
```

Migration

- übernimmt später CREATE/ALTER TABLE

```
$ rails generate scaffold lectures
  titel:string sws:integer vorl_nr:integer
  professor:references
```

<=>

```
$ rails generate scaffold lectures
  titel:string sws:integer vorl_nr:integer
  professor_id:integer
```

Migration

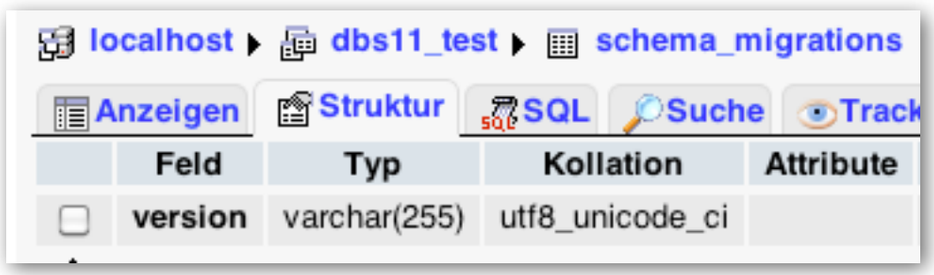
- sind eigene Klassen in

  `meine_applikation/db/migrate`

- können automatisiert ausgeführt werden

  `rake db:migrate VERSION=123`

- Datenbank kennt aktuelle Version



Model

- automatisch generierte Model-Klassen

```
class Lecture < ActiveRecord::Base
  belongs_to :professor
end
```

```
$ rails generate scaffold lectures
  titel:string sws:integer vor_nr:integer
  professor:references
```

- Obacht: Klasse `Professor` wird nicht angepasst

Controller & Views

- Controller und Views für CRUD-Operationen
- betrachtet Entität als Ressource
- REST-Operationen

  GET
  POST
  PUT
  DELETE

- Bietet HTML- und XML-Zugriffsmöglichkeiten

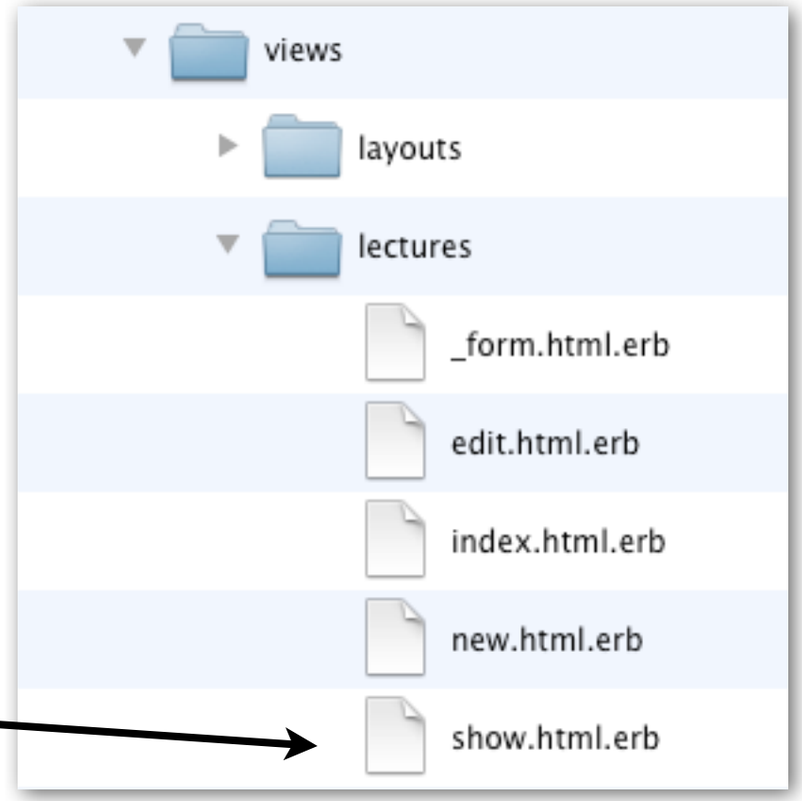
Controller & Views

```
class LecturesController < ApplicationController

def show
    @lecture = Lecture.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.xml  { render :xml => @lecture }
    end
  end

  ...

end
```

Controller & Views

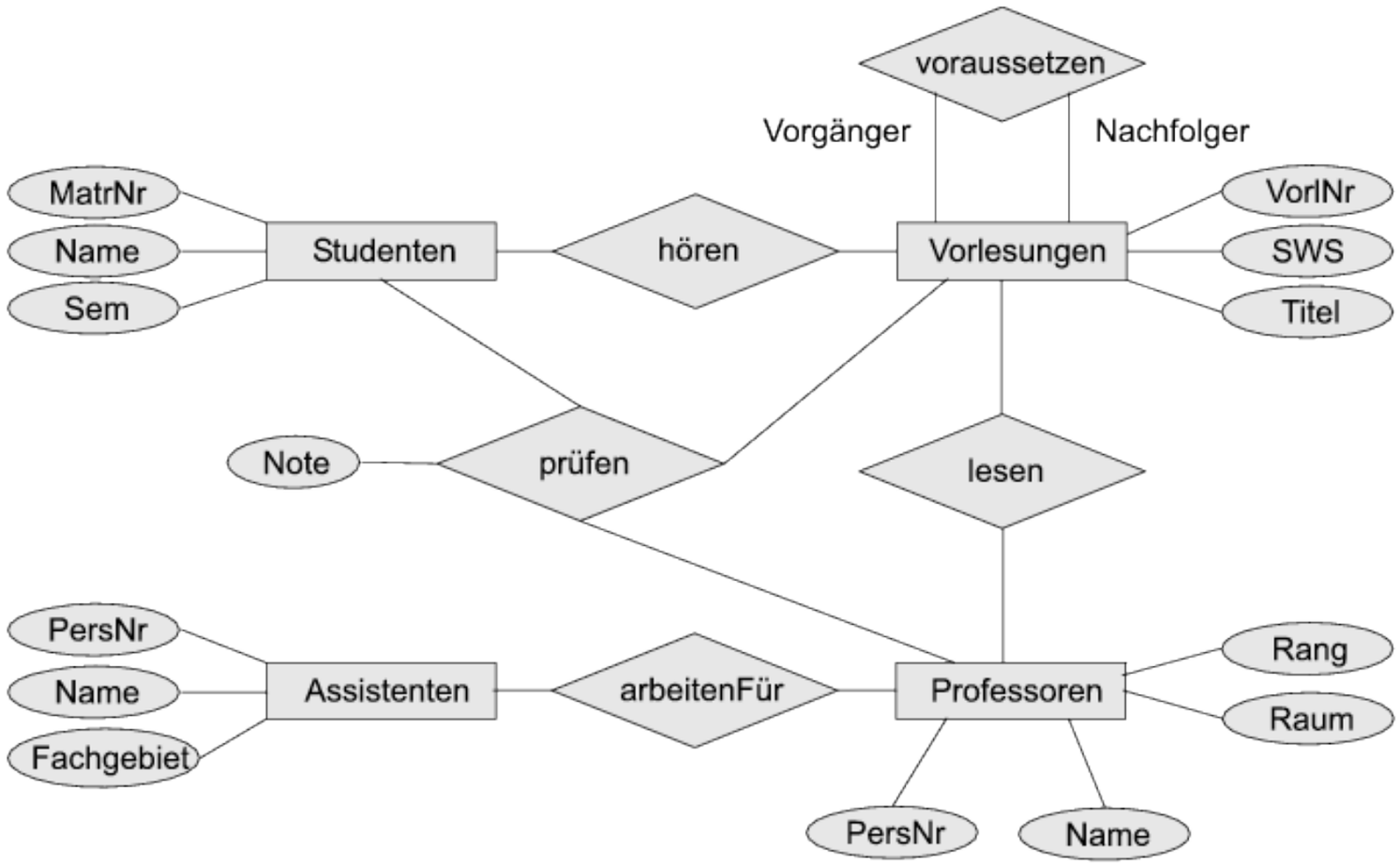
- Erinnerung:

```
def show
   ...
end

render :action => „show“
```



- HTML-Oberflächen für CRUD-Operationen

Neue Applikation von 0

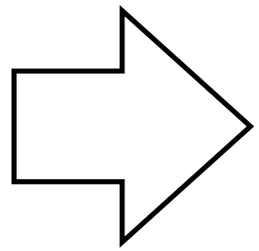


Neue Applikation von 0

Demo

Scaffolds: Wiederholung

- `rails generate scaffold lectures`
- erzeugt:
  - Migration
  - Model
  - Controller
  - Views
  - Tests



CREATE TABLE ...

Lecture.new

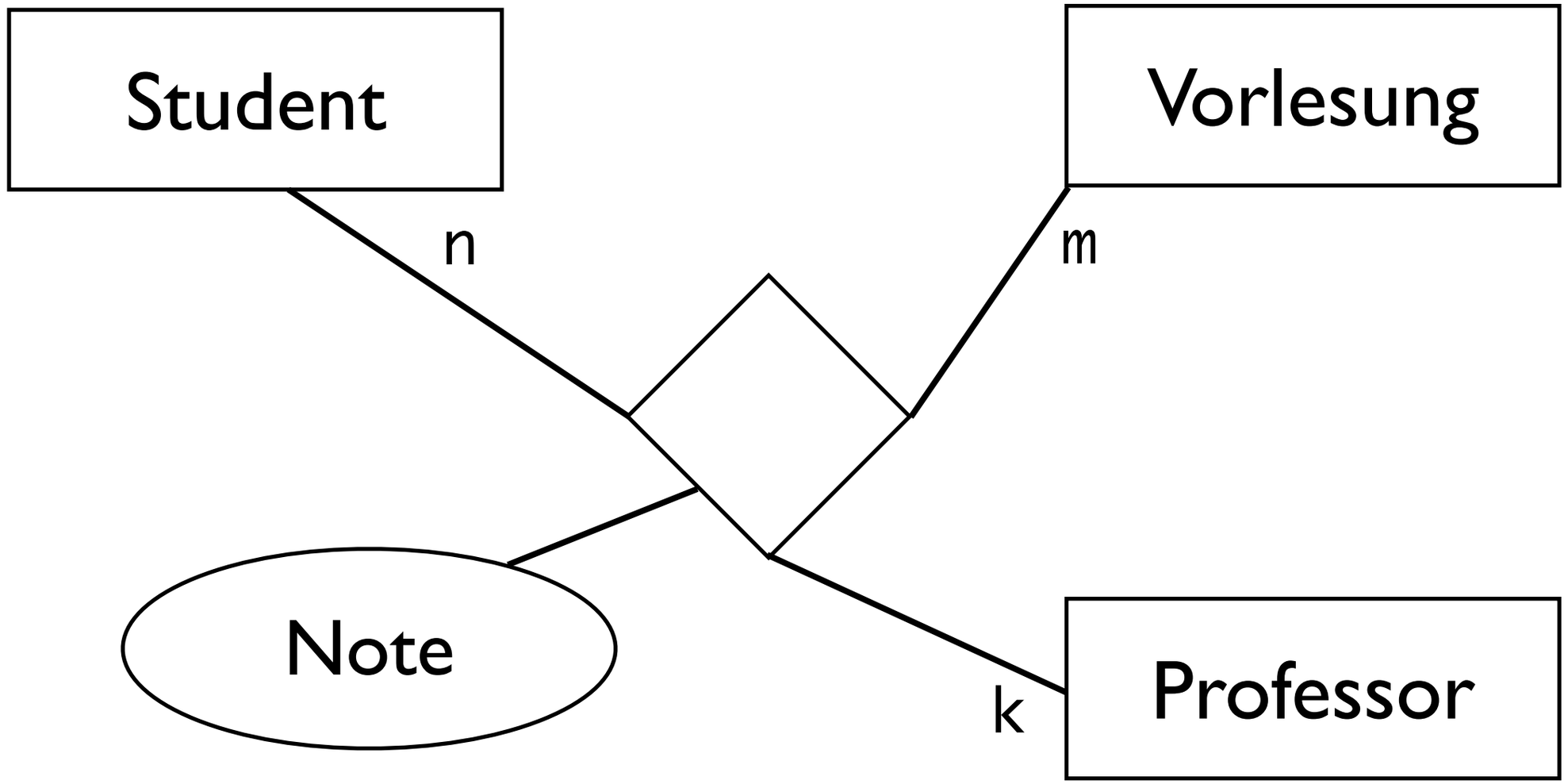
http://.../lecture/show

show.html.erb

...

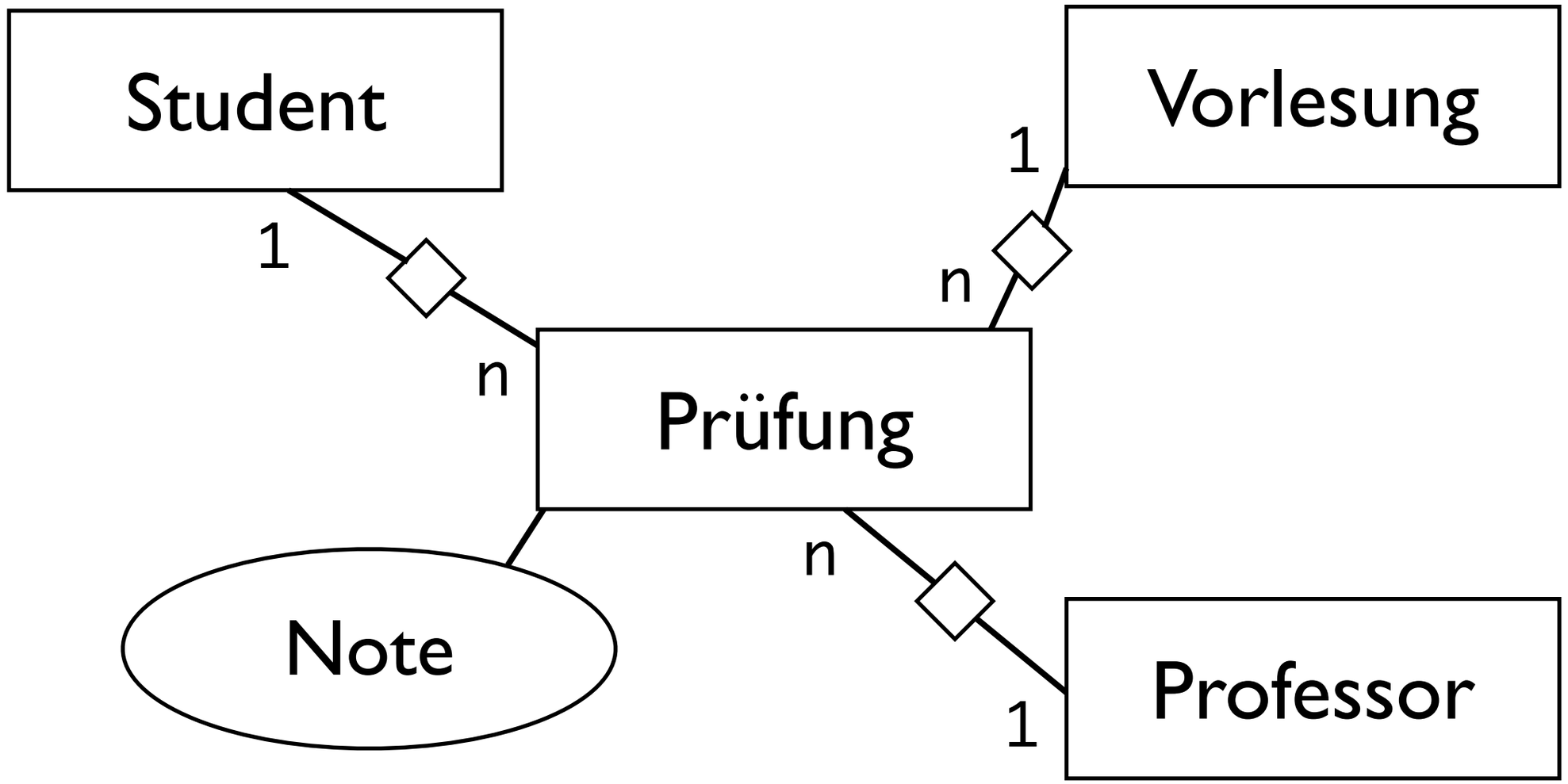
Fortgeschrittene ORM-Techniken

- mehrwertige Relationen und Relationen mit Attributen



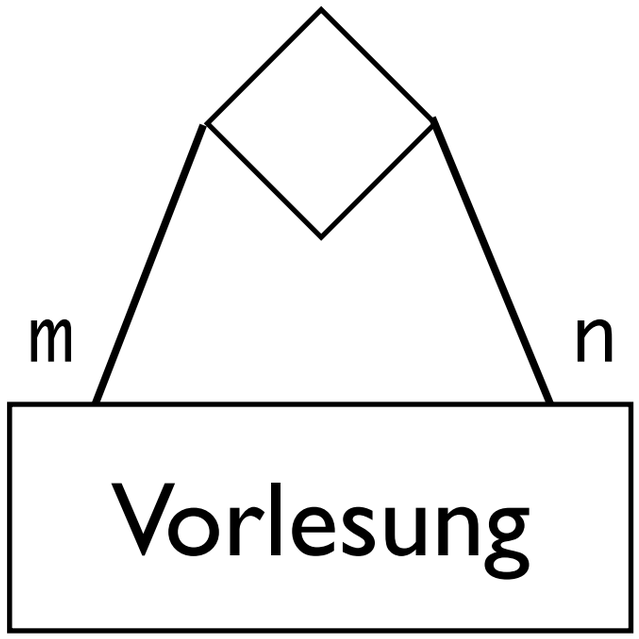
Fortgeschrittene ORM-Techniken

- mehrwertige Relationen und Relationen mit Attributen



Fortgeschrittene ORM-Techniken

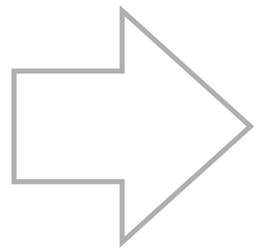
- Entitäten mit N:M-Selbstreferenz




```
has_and_belongs_to_many :vorgaenger,
:class_name => "Lecture", :join_table =>
"lectures_lectures", :foreign_key =>
"nachfolger_id", :association_foreign_key =>
"vorgaenger_id"
```

Scaffolds: Wiederholung

- `rails generate scaffold lectures`
- erzeugt:
  - Migration
  - Model
  - Controller
  - Views
  - Tests



CREATE TABLE ...

Lecture.new

http://.../lecture/show

show.html.erb

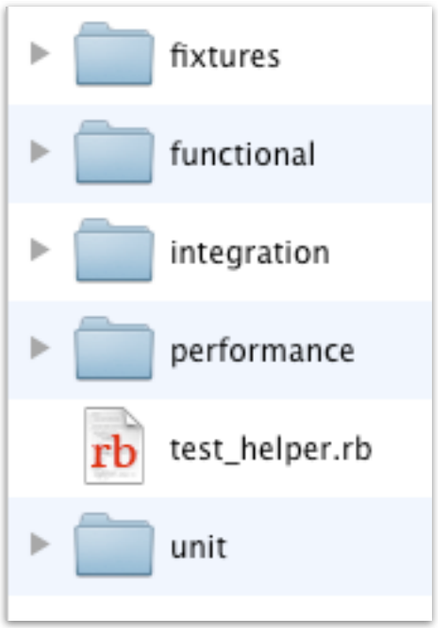
...

Testen in Rails

- Test-Driven-Development
- Kernidee von Rails
- integrierte Tests für
  - Models (Unit-Tests)
  - Controller (Functional Tests)
  - Views (Functional Tests)
  - Controller-Interaktion (Integration Tests)

Testen in Rails

- Verzeichnis `test`



← Testdaten
← Tests für Controller/Views
← Tests für Controller-Interaktion
← Tests für Models

Einfacher Unit-Test

- unit/student_test.yml

```
require 'test_helper'

class StudentTest < ActiveSupport::TestCase
  # Replace this with your real tests.
  test "the truth" do
    assert true
  end
end
```

Einfacher Unit-Test

```
require 'test_helper'

class StudentTest < ActiveSupport::TestCase

  test "valid semester" do
      s = Student.new
      s.name = "Willi"
      s.matr_nr = 123
      s.semester = -1  #fehlerhafte daten
      assert !s.save, "Neg. Semester erlaubt"

  end
end
```

Einfacher Unit-Test

- Test ausführen

```
$ ruby -Itest test/unit/student_test.rb
```

für test_helper.rb

Test(s)

- Lädt Testdaten in Datenbank
- Führt alle Tests aus

Einfacher Unit-Test

- Ergebnis:

```
Loaded suite test/unit/student_test
Started
F
Finished in 0.105289 seconds.

  1) Failure:
test_valid_semester(StudentTest)
[test/unit/student_test.rb:11]:
Neg. Semester erlaubt

1 tests, 1 assertions, 1 failures,
0 errors, 0 skips
```

Test-Driven-Development

- Erst Test schreiben, dann Funktionalität entwickeln
- Beispiel: Validierung von Daten eines Models
- app/models/student.rb

```
validates :semester,
          :presence => true,
          :inclusion => 1..25
```

- Obacht: Niemand verhindert INSERT INTO ...

Einfacher Unit-Test

- Ergebnis:

```
Loaded suite test/unit/student_test
Started
.
Finished in 0.028667 seconds.

1 tests, 1 assertions, 0 failures,
0 errors, 0 skips
```

Fixtures

- Nicht auf Produktivdaten testen
- Fixtures = Testdaten
- YAML oder CSV
- Test-Datenbank in database.yml konfigurieren
- ggf. mit `rake db:test:prepare` initialisieren
- vor dem Testen automatisch in die Testdatenbank geladen

Fixtures

- `fixtures/students.yml`

```
one:
  matr_nr: 1
  name: MyString
  semester: 1

two:
  matr_nr: 1
  name: MyString
  semester: 1
```

Fixtures

- fixtures/students.yml

```
willi:
  matr_nr: 123
  name: Willi Wacker
  semester: 5
  lectures: infob
```

Fixtures

- fixtures/lectures.yml

```
infoa:
  titel: Info A
  vorl_nr: 1
  sws: 9

infob:
  titel: Info B
  vorl_nr: 2
  sws: 9
  vorgaenger: infoa
```

Unit-Test mit Testdaten

- models/student.rb

```
def fehlende_vl

  erg = []
  self.lectures.each do |l|
    erg = erg + (l.vorgaenger - self.lectures)
  end

  return erg.uniq

end
```

Unit-Test mit Testdaten

```
require 'test_helper'

class StudentTest < ActiveSupport::TestCase

  test "vl_ohne_vorgaenger funktion" do
        s = students(:willi)
        assert s.fehlende_vl.include?
               (lectures(:infoa)),
               "Willi sollte Info A fehlen"

  end
end
```

Unit-Test mit Testdaten

- Ergebnis:

```
Loaded suite test/unit/student_test
Started
..
Finished in 0.097540 seconds.

2 tests, 2 assertions, 0 failures,
0 errors, 0 skips
```

Weitere Informationen

- Mehr zum Thema *Testen mit Rails*
- Übungsblatt und Übung
- http://guides.rubyonrails.org/testing.html

Ein „echtes“ Beispiel

- Was fehlt eigentlich noch?
- „echte“ Queries
- Datenbank-Performance ausnutzen

- IMDB-Datenbank:
  O(n * log n) bei 60 Millionen Einträgen...
- ActiveRecord-Query-Interface

ActiveRecord

- Datenabfragen mit eigenem Query-Interface

```
Movie.first
Movie.find(123)
Movie.where(:name => „abc“).where(:year => 1942)
Movie.where(„name = ?“, @name)
Movie.order(„name“)

offset, limit, group, having, ...
```

- Benötigt passende Model-Klasse

ActiveRecord

- Direkte Anfragen an die Datenbank

```
ActiveRecord::Base.connection.

select_all(„SELECT * FROM movies ...“)
select_one(„SELECT * FROM movies ...“)
...
```

- Liefert einen Array mit Hashes oder Hashes

```
h = [{:title => „Casablanca“, :id => 1}, {...}]

h[0][:title]
```

Erinnerung

- models/student.rb

```
def fehlende_vl

  erg = []
  self.lectures.each do |l|
    erg = erg + (l.vorgaenger - self.lectures)
  end

  return erg.uniq

end
```

Erinnerung

- models/student.rb

```
def fehlende_vl
  Lecture.find_by_sql("SELECT l.* FROM lectures
  l, lectures_students ls, lectures_lectures ll
  WHERE ls.lecture_id = ll.nachfolger_id and
  ls.student_id = " + self.id.to_s + " and
  ll.vorgaenger_id = l.id and ll.vorgaenger_id
  not in (SELECT ls2.lecture_id FROM
  lectures_students ls2 WHERE ls2.student_id =
  " + self.id.to_s + ")")
end
```

Erinnerung

- Ergebnis:

```
Loaded suite test/unit/student_test
Started
..
Finished in 0.097540 seconds.

2 tests, 2 assertions, 0 failures,
0 errors, 0 skips
```

Zusammenfassung

- Datenbankabstraktion mit ORM und ActiveRecord
- Applikationsdesign mit Model-View-Controller
- Don‘t Repeat Yourself
- Convention over Configuration
- Scaffolds & Migrations
- Bestehende Datenbanken und neue Applikationen
- Tests

Literatur und weitere Informationen

- Agile Web Development with Rails.
  Thomas, Dave; Heinemeier Hansson, David.
  (Obacht: Fourth Edition für Rails 3, Rest veraltet!)
- Rails API:
  http://api.rubyonrails.org/
- Ruby-Dokumentation:
  http://corelib.rubyonrails.org/
- Rails Guides:
  http://guides.rubyonrails.org/