Tecnologie per lo sviluppo di applicazioni Internet

Dott. Fabio Fioravanti

fioravanti@sci.unich.it

http://www.sci.unich.it/~fioravan

Corso di Laurea Specialistica in Economia Informatica
Facoltà di Economia
UNICH - AA 2008/2009

Programma

- Ripasso nozioni di base
- Model View Controller
- Ruby / Java / PHP on Rails
- AJAX (Asynchronous JavaScript and XML)

- Accessibilità
- Usabilità
- Aspetti normativi ed organizzativi

Materiale di studio

- Lucidi del corso
- Guide e manuali
- Articoli, libri, siti web e materiale reso disponibile durante il corso

Info

- Lezioni
  - Mer, Gio h10:30-12:30
- Ricevimento
  - su appuntamento
- Modalità d’esame
  - Progetto
  - Orale

Prerequisiti

- Saper programmare utilizzando linguaggi procedurali e orientati agli oggetti
- Tecnologie web
  - (X)HTML, CSS, Javascript
- Basi di dati
- Saper leggere documenti in inglese

Model-view-controller (MVC)

- **Model-view-controller** (**MVC**) is an architectural pattern used in software engineering.
- separate data (model) and user interface (view) concerns, so that changes to the user interface will not affect data handling, and that the data can be reorganized without changing the user interface.
- decoupling data access and business logic from data presentation and user interaction, by introducing an intermediate component: the controller.

- Increase flexibility and reuse

MVC

- Model
  - The **domain**-specific representation of the information on which the application operates.
  - Domain logic adds meaning to raw data (e.g., calculating whether today is the user's birthday, or the totals, taxes, and shipping charges for shopping cart items).
  - Many applications use a persistent storage mechanism (such as a database) to store data.
  - MVC does not specifically mention the data access layer because it is understood to be underneath or encapsulated by the Model.

MVC

- View
  - Renders the model into a form suitable for interaction, typically a user interface element.
  - Multiple views can exist for a single model for different purposes.
  - Has the responsibility of mantaining consistency of presentation when data change.
    - Push model – the view registers itself to be notified of changeds in the model
    - Pull model – the view “calls” the model for up-to-date data

MVC

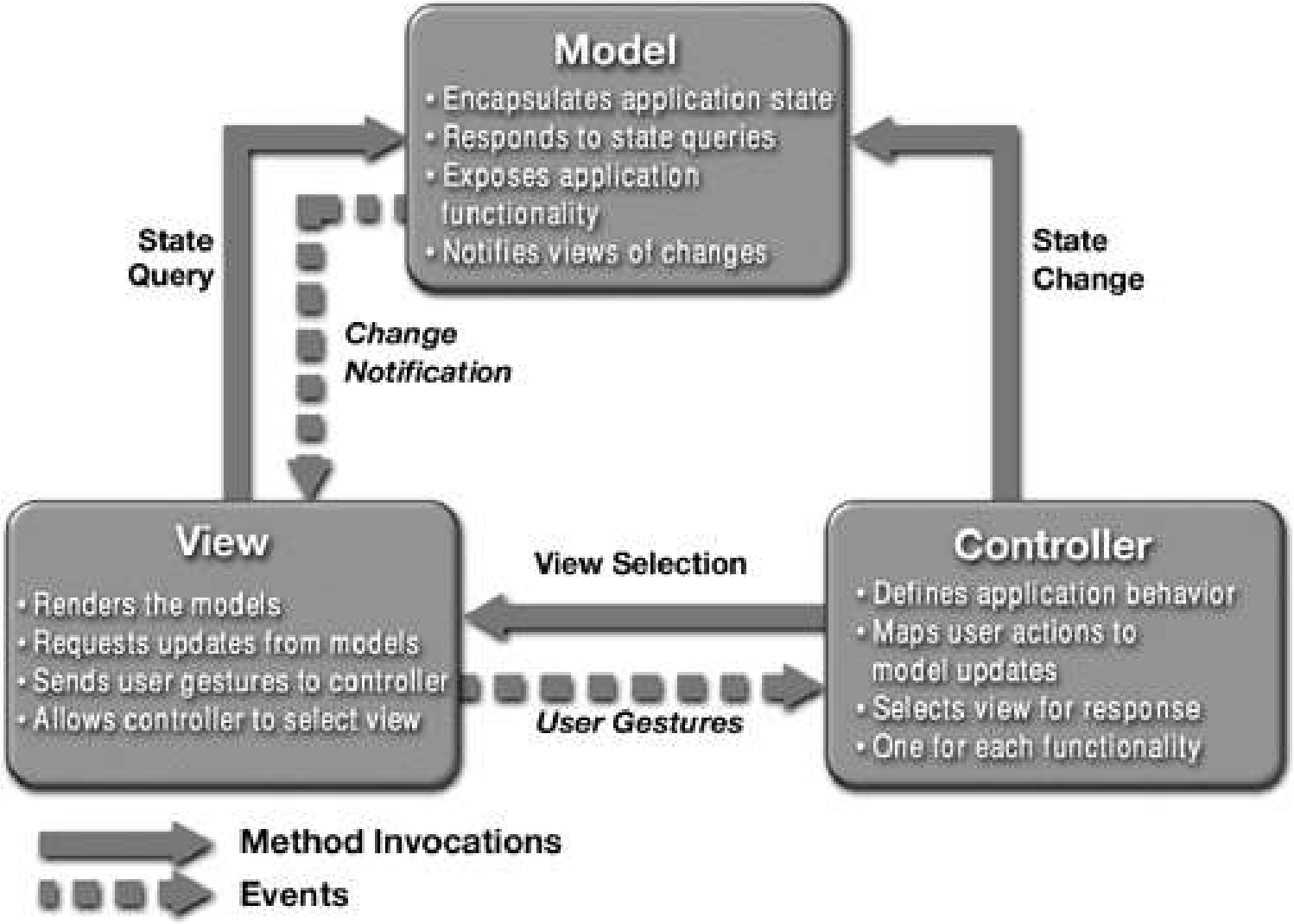
- Controller
  - Processes and responds to events, typically user actions
  - may invoke changes on the model
  - Selects a view to be displayed

MVC in web applications

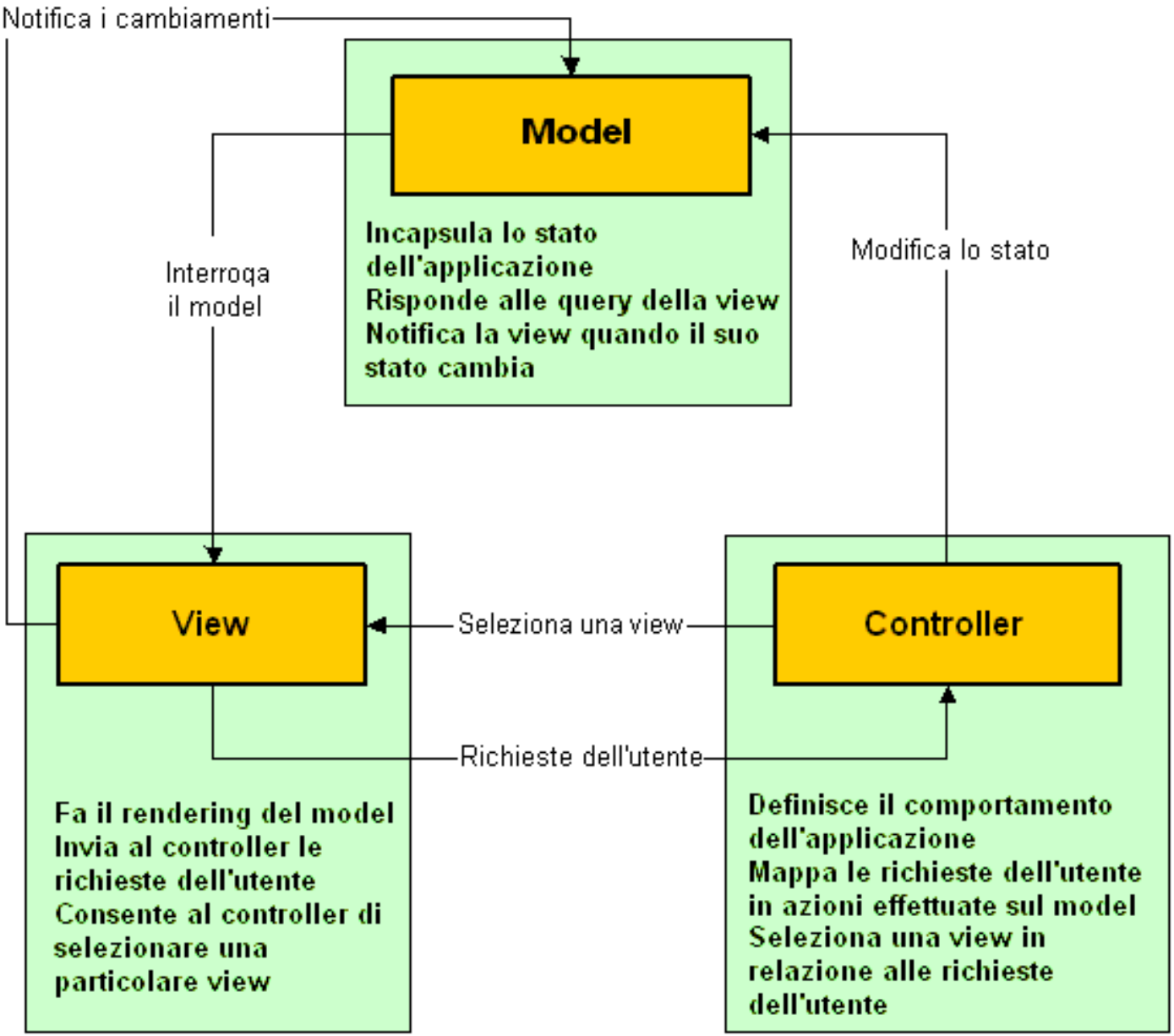
- the view is the actual HTML page,
- the controller is the code that gathers dynamic data and generates the content within the HTML.
- the model is represented by the actual content, usually stored in a database or XML files.

MVC – (typical) control flow

- The user interacts with the user interface in some way (*e.g.*, presses a button).
- A controller handles the input event from the user interface, often via a registered handler or callback.
- The controller accesses the model, possibly updating it in a way appropriate to the user's action
- A view uses the model (indirectly) to generate an appropriate user interface (*e.g.*, the view produces a screen listing the shopping cart contents). The view gets its own data from the model. The model has no direct knowledge of the view.
- The user interface waits for further user interactions, which begins the cycle anew.

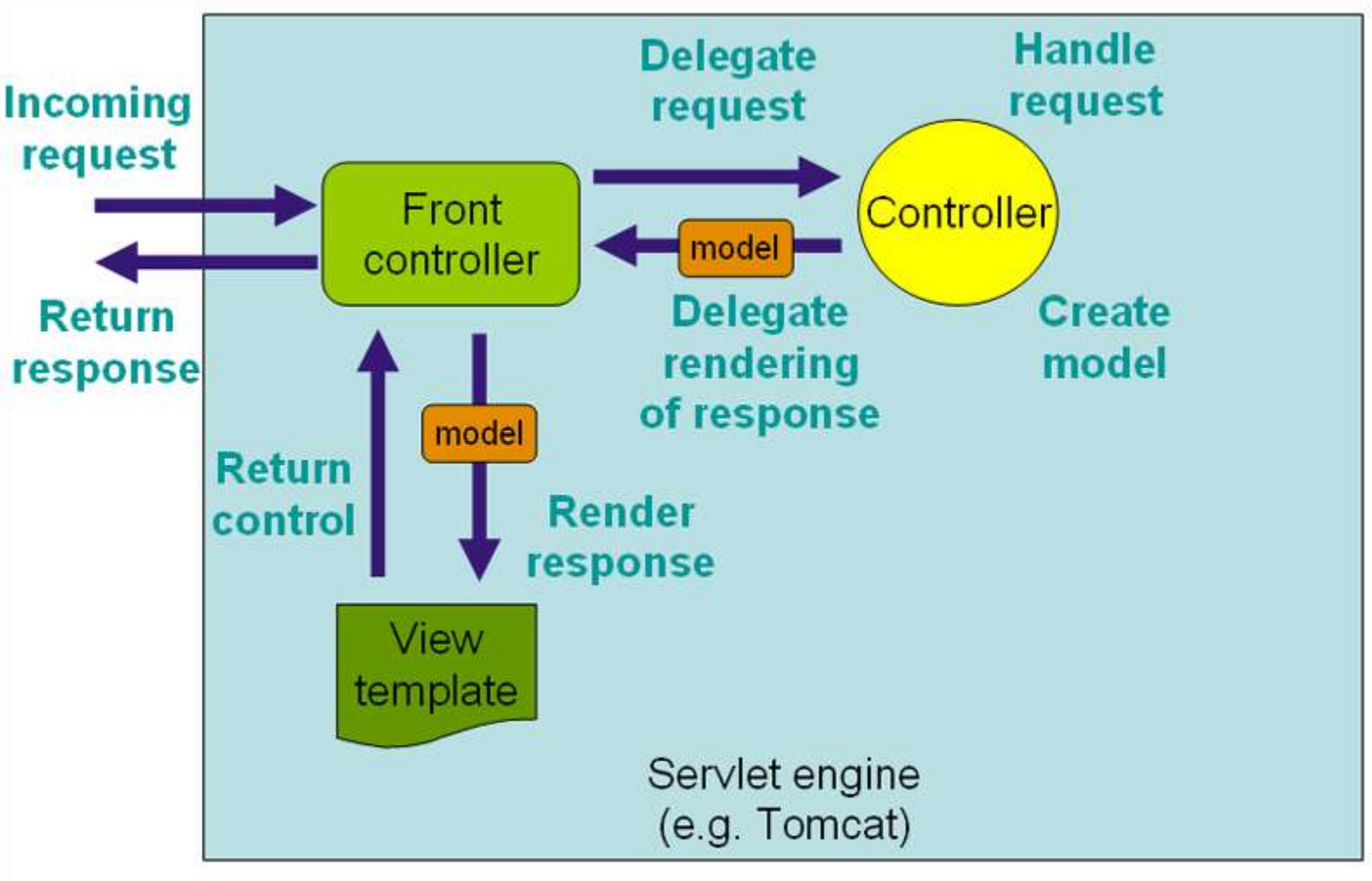


http://java.sun.com/developer/technicalArticles/javase/mvc/images/Figure1.gif



http://java.sun.com/developer/technicalArticles/javase/mvc/images/Figure1.gif

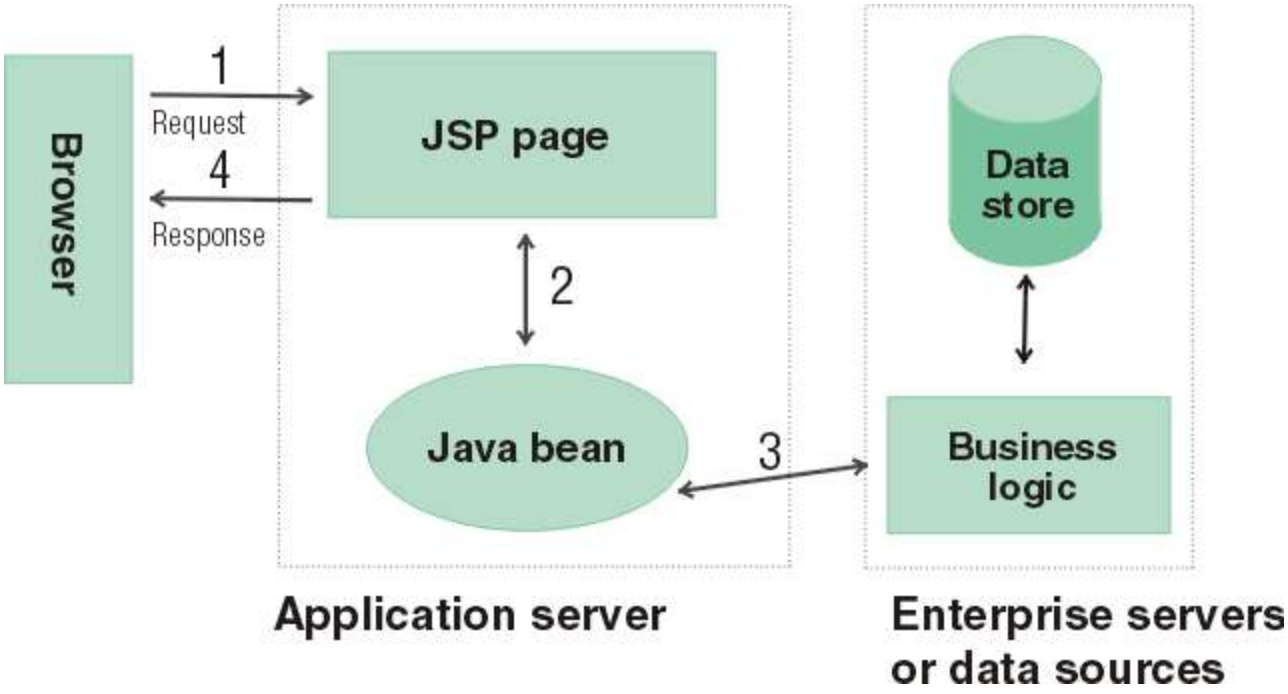
Struts MVC



http://www.twoqubed.com/svn/presentations/spring-mvc/trunk/docs/mvc.png

MVC Model 1

- In the JSP Model 1 architecture, the JSP page alone processes the incoming request and replies to the client



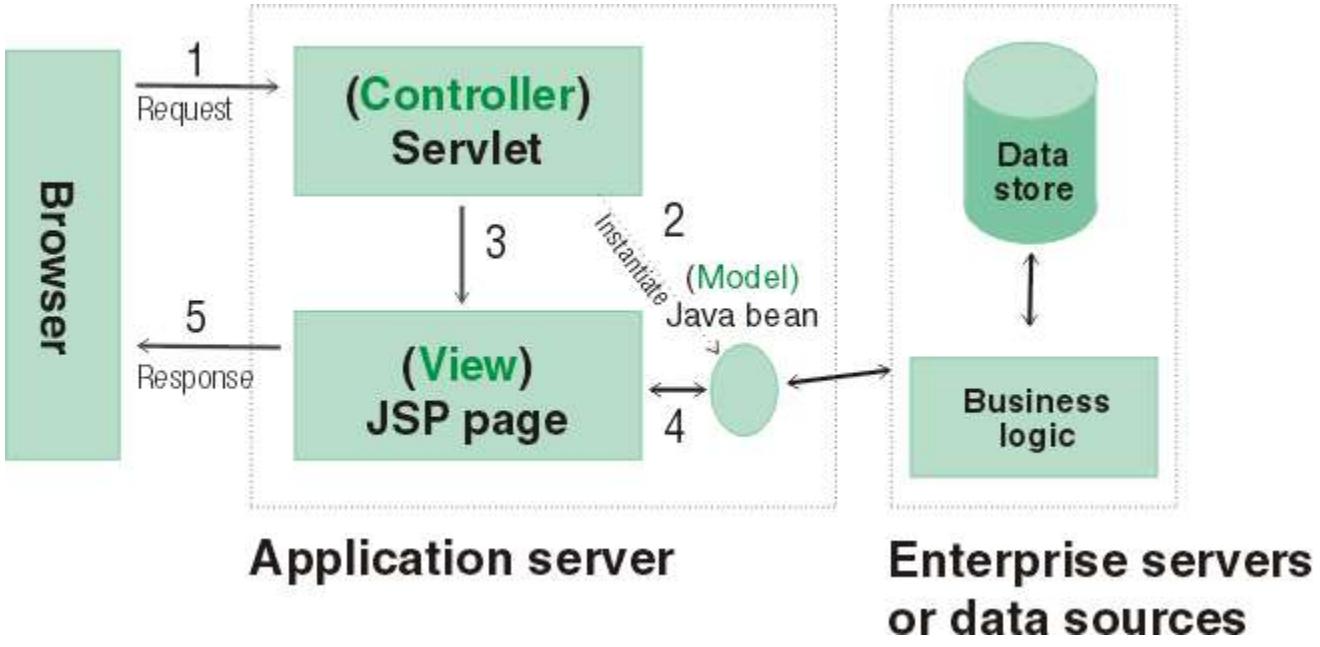
http://publib.boulder.ibm.com/infocenter/iadthelp/v7r0/topic/com.ibm.etools.struts.doc/topics/cstrdoc001.html

MVC Model 1

1. The browser sends a request to a JSP page.
2. The JSP page communicates with a Java bean.
3. The Java bean is connected to a database.
4. The JSP page responds to the browser.

MVC Model 2

- the servlet processes the request, creates any beans or objects used by the JSP file, and forwards the request



http://publib.boulder.ibm.com/infocenter/iadthelp/v7r0/topic/com.ibm.etools.struts.doc/topics/cstrdoc001.html

MVC Model 2

1. The browser sends a request to a servlet.
2. The servlet instantiates a Java bean that is connected to a database.
3. The servlet communicates with a JSP page.
4. The JSP page communicates with the Java bean.
5. The JSP page responds to the browser.

MVC Model 1 and 2

- Model 1
  - Simple web app
  - Quick prototyping
  - View and controller being done by the same team
- Model 2
  - Complex web app
  - Creating an application to be modified and maintained
  - View and controller being done by different teams

Ruby on Rails

A gentle introduction

Ruby on Rails

- Ruby on Rails is a free web application framework
- aims to increase the speed and ease with which database-driven web sites can be created
- offers skeleton code frameworks (scaffolding) from the outset.
- Often shortened to Rails, or RoR
- written in the Ruby programming language
- applications using the Rails framework are developed using the Model-View-Controller design pattern.

"Don't repeat yourself"

- information is located in a single, unambiguous place.
- For example, using ActiveRecord, the developer does not need to specify database column names in class definitions. Instead, Ruby can retrieve this information from the database.

- Es: una tabella “prodotto” contiene un campo “quantita” di tipo intero, non negativo
  - il vincolo viene specificato una sola volta nell’applicazione.

"Convention over Configuration"

- a developer only needs to specify unconventional aspects of the application.
- For example, if there's a class Sale in the model, the corresponding table in the database is called sales by default.
- only if one deviates from this convention, such as calling the table "products_sold", there is the need to write code regarding these names.

History

- Ruby on Rails was extracted by David Heinemeier Hansson from his work on Basecamp, a project-management tool by the web company 37signals.
- First released to the public in July 2004.
- Apple will ship Ruby on Rails with Mac OS X v10.5 Leopard, scheduled for release in October 2007.

Scaffolding

- Scaffolding is a technique which allows to
  - generate code that the application can use to create, read, update and delete database entries (CRUD)
  - Dynamic (run-time)
  - Generative (generate files)

object-relational mapping

- converting data between incompatible type systems in databases and object-oriented programming languages
- Map database tables to classes
- Map table rows to objects
- Map table columns to attributes
- Methods for
  - Selecting
  - Updating
  - Inserting
  - Deleting
- No SQL for simple operations

object-relational mapping

- SQL
- CREATE TABLE Product (
  id INTEGER,
  name VARCHAR(100),
  price DOUBLE,
  stock INTEGER)

- JAVA
- public class Product {
  private String name;
  private Double price;
  private int stock;
  ... }
- Ruby
- class product <
  ActiveRecord::Base
  end

NO ORM vs ORM

- <DB connect>
- SELECT * FROM user WHERE id=97
- UPDATE user SET age=18 WHERE id=97

- user = User.find(97)
- user.age = 18
- user.save

Technical features

- Rails uses the Model-View-Controller (MVC) architectural pattern
- Rails provides 'out of the box' scaffolding
- WEBrick web server for development (not for production)
- Rake build system
- extensive use of the JavaScript libraries Prototype and Script.aculo.us for Ajax and its graphical interface.
- Databases
  - MySQL, PostgreSQL, SQLite, DB2, Oracle and SQL Server.
- web services
  - Rails initially supported lightweight SOAP;
  - later it was replaced by RESTful web services.

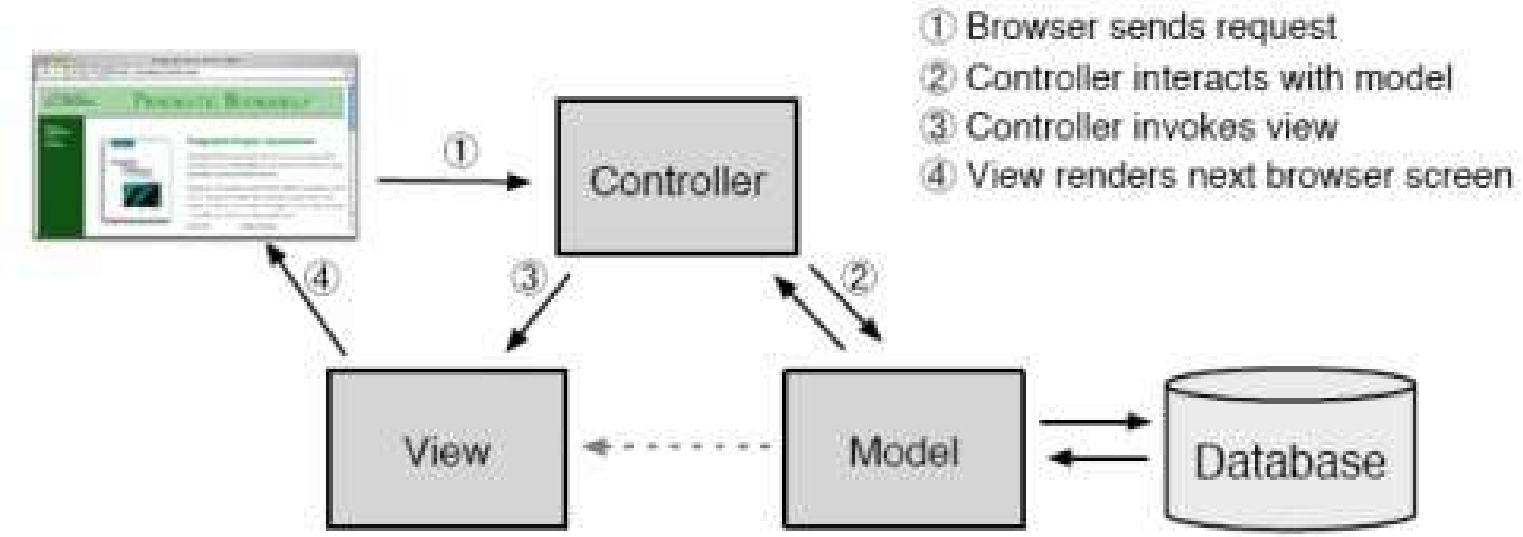
REST

- Representational State Transfer
  - Uno stile architetturale per web applications e web services
- Fondamentale è il concetto di risorsa
  - Manipolata tramite metodi di HTTP
  - In contrasto con RPC e SOAP
- Esempio
  - RPC: getUser(12)
  - REST: GET http://example.com/users/12
  - RPC:
    exampleAppObject = new ExampleApp('example.com:1234')
    exampleAppObject.removeUser('001')
  - REST:
    userResource = new Resource('http://example.com/users/001')
    userResource.delete()

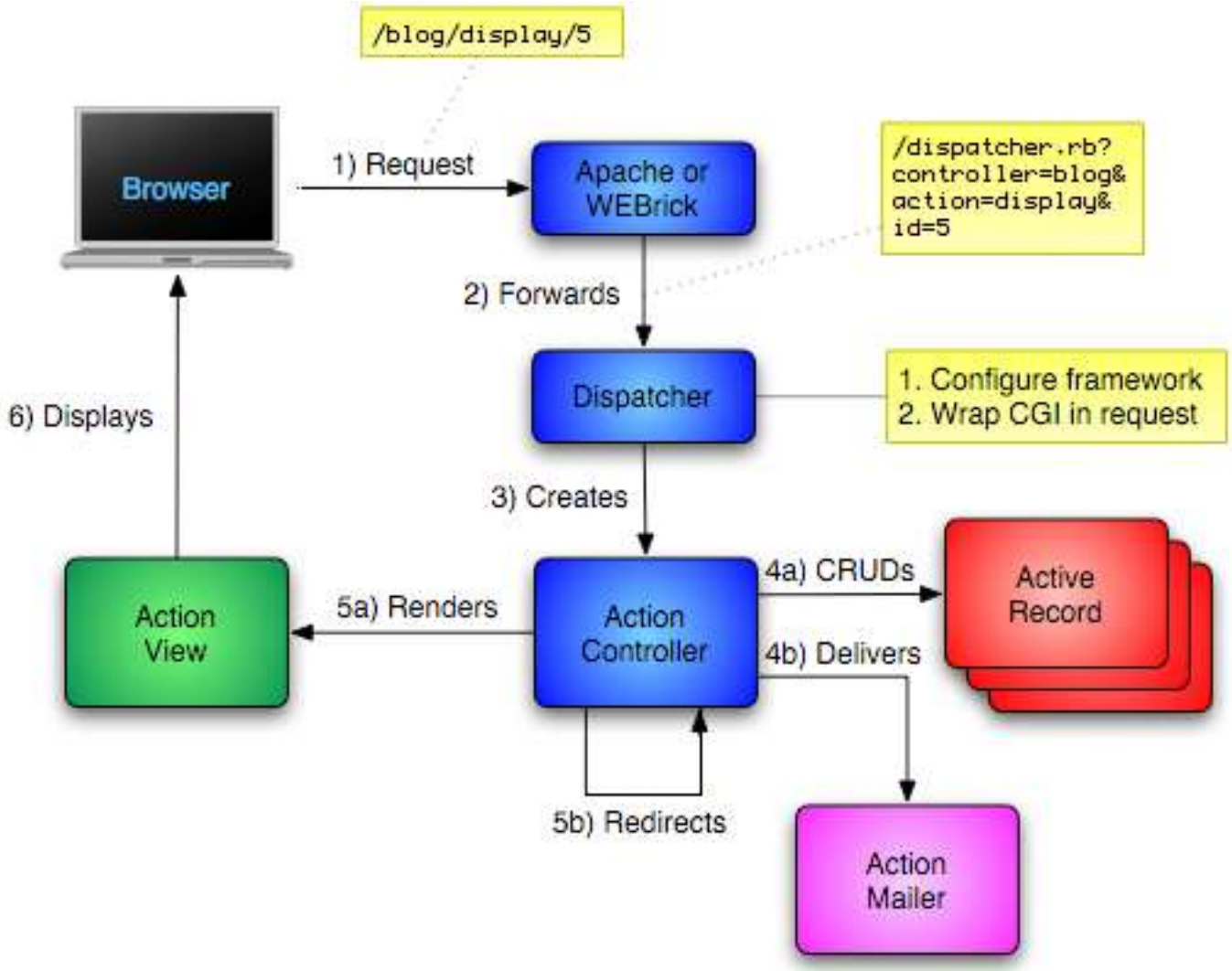
Alternatives

- Java
  - Appfuse (Struts 2, Spring, Hibernate, Tapestry), OpenSails
  - Grails (Groovy language)
- PHP
  - CakePHP, Symfony, Akelos, CodeIgniter
- Python
  - Django, TurboGears

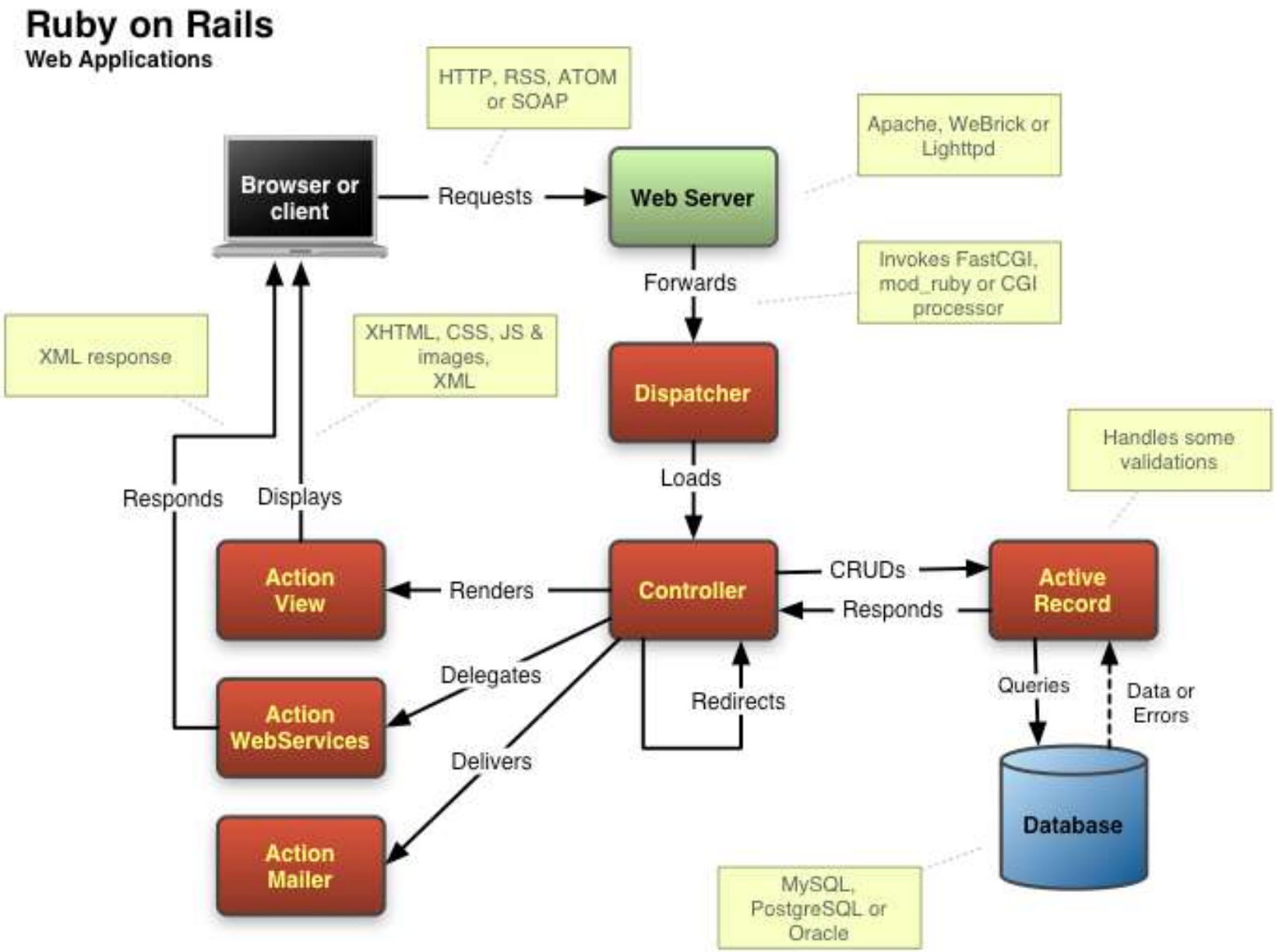
MVC in Rails



http://alexmrg.files.wordpress.com/2006/12/mvc.JPG



http://www.intertwingly.net/slides/2005/fosssl/mvc.png



http://dotsx.usc.edu/rails2.png

Installing RoR

- RubyonRails.org
- InstantRails
  - http://instantrails.rubyforge.org

Installation on Win XP

- Download and install Ruby 1.8.6
- Connect to the Internet
- Remember to properly configure your firewall
- Download and install RubyGems (not needed in our case: already installed!)
- C:\ruby>gem install rails --include-dependencies
  Bulk updating Gem source index for: http://gems.rubyforge.org
  Successfully installed rails-1.2.3
  Successfully installed activesupport-1.4.2
  Successfully installed activerecord-1.15.3
  Successfully installed actionpack-1.13.3
  Successfully installed actionmailer-1.3.3
  Successfully installed actionwebservice-1.2.3
  Installing ri documentation for activesupport-1.4.2...
  Installing ri documentation for activerecord-1.15.3...
  Installing ri documentation for actionpack-1.13.3...
  Installing ri documentation for actionmailer-1.3.3...
  Installing ri documentation for actionwebservice-1.2.3...
  Installing RDoc documentation for activesupport-1.4.2...
  Installing RDoc documentation for activerecord-1.15.3...
  …

Installation

- C:\railsapps\simpleapp>gem install sqlite3-ruby

Select which gem to install for your platform (i386-mswin32)
 1. sqlite3-ruby 1.2.1 (mswin32)
 2. sqlite3-ruby 1.2.1 (ruby)
 3. sqlite3-ruby 1.2.0 (mswin32)
 4. sqlite3-ruby 1.2.0 (ruby)
 5. Skip this gem
 6. Cancel installation
> 1
Successfully installed sqlite3-ruby-1.2.1-mswin32
Installing ri documentation for sqlite3-ruby-1.2.1-mswin32...
Installing RDoc documentation for sqlite3-ruby-1.2.1-mswin32...

Build an app

- C:\>mkdir railsapps
- C:\>cd railsapps
- C:\railsapps>rails simpleapp
  create
  create app/controllers
  create app/helpers
  create app/models
  create app/views/layouts
  create config/environments
  create components
  create db
  create doc
  …

Directory structure

- simpleapp/
  - README
    - Installation and usage information
  - Rakefile
    - Manage documentation, testing, DB migration
    - rake --tasks
  - app/
    - Model, View, and Controller files
  - components/
    - Reusable components… are being deprecated

Directory structure

- simpleapp/
  - config/
    - database.yml -> database parameters
    - environment.rb
    - development.rb, test.rb, production.rb
  - db/
    - Schema and migration information
  - doc/
    - Autogenerated documentation
    - rakedoc:app

Directory structure

- simpleapp/
  - lib/
    - Shared code not fitting in MVC
    - Ex: third-party pdf library
  - log/
    - Logfiles produced by your application
    - development.log, test.log, production.log
    - Timing statistics,cache information, db queries
  - public/
    - Web-accessible directory. Your application runs from here

Directory structure - scripts

- simpleapp/script/
  - breakpointer
  - console
  - generate
    - model
    - controller
    - scaffold
  - plugin
    - manage plugins
  - process/
    - control running app
  - performance/
    - benchmarking, profiler

Directory structure

- simpleapp/
  - test/
    - Unit, functional, and integration tests, fixtures, and mocks
  - tmp/
    - Runtime temporary files
  - vendor/
    - Imported code
    - plugins/

Naming conventions

- Convention over configuration! Conventions can be overridden
- variables are lowercase
  - words separated by underscore “_”
  - ex: line_item
- classes
  - no underscores, mixed-case, CamelCased
  - ex: LineItem
- methods
  - ex: find, new, save, destroy
- database tables
  - lowercase
  - plural names (in English): sounds good in conversation
  - ex: line_items table and app/model/line_item.rb file

Model Naming

- Table
  - line_items
- Class
  - LineItem
- File
  - app/models/line_item.rb

Controller Naming

- URL
  - http://.../store/list
- Class
  - StoreController
- File
  - app/controllers/store_controller.rb
- Method
  - list

View Naming

- URL
  - https://.../store/list
- File
  - app/views/store/list.rhtml (or .rxml, .rjs)
- Helper module
  - StoreHelper
- File
  - app/helpers/store_helper.rb

Overriding conventions

- ActiveRecord::Base.pluralize_table_names= false
- class Sheep<ActiveRecord::Base

  set_table_name "sheep" #Not "sheeps“

  end

Database configuration

- in config/database.yml
- development:
  adapter: mysql
  database: todos
  username: root
  password: root
  host: localhost

Start the server

- C:\railsapps>cd simpleapp
- C:\railsapps\simpleapp>ruby script/server

  => Booting WEBrick...

  => Rails application started on http://0.0.0.0:3000

  => Ctrl-C to shutdown server; call with --help for options

  [2007-10-02 17:28:01] INFO WEBrick 1.3.1

  [2007-10-02 17:28:01] INFO ruby 1.8.6 (2007-03-13) [i386-mswin32]

  [2007-10-02 17:28:03] INFO WEBrick::HTTPServer#start: pid=2292 port=3000
- Remember to properly configure your firewall

Bootstrapping

- Visit http://localhost:3000/
- Create your databases and edit config/database.yml
  - Rails needs to know your login and password.
- Use script/generate to create your models and controllers
  - To see all available options, run it without parameters.
- Set up a default route and remove or rename this file
  - Routes are setup in config/routes.rb.

Create “categories” table

- CREATE TABLE `categories` (
  `id` smallint(5) unsigned NOT NULL auto_increment,
  `category` varchar(20) NOT NULL,
  `created_on` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `updated_on` timestamp NOT NULL default '0000-00-00 00:00:00',
  PRIMARY KEY (`id`),
  UNIQUE KEY `category_key` (`category`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='List of categories'
  AUTO_INCREMENT=6 ;
- INSERT INTO `categories` (`id`, `category`, `created_on`, `updated_on`) VALUES
  (1, 'Home & Family', '0000-00-00 00:00:00', '0000-00-00 00:00:00'),
  (2, 'Business', '0000-00-00 00:00:00', '0000-00-00 00:00:00'),
  (4, 'Rails documentation', '0000-00-00 00:00:00', '0000-00-00 00:00:00'),
  (5, 'Community Council', '0000-00-00 00:00:00', '0000-00-00 00:00:00');

Notes

- every table should have a primary key named “id”
- links to other tables with <targettable>_id
- rails automatically mantains
  - created_at / created_on
  - updated_at / updated_on

FourDays on Rails

- ruby script/generate controller category
  - exists app/controllers/
    exists app/helpers/
    create app/views/category
    create test/functional/
    create app/controllers/category_controller.rb
    create test/functional/category_controller_test.rb
    create app/helpers/category_helper.rb
- ruby script/generate model category
  - create app/models/
    create test/unit/
    create test/fixtures/
    create app/models/category.rb
    create test/unit/category_test.rb
    create test/fixtures/categories.yml
    exists db/migrate
    create db/migrate/001_create_categories.rb

Directory structure… updated

- simpleapp/app/
  - controllers/
    - application.rb
    - category_controller.rb
  - helpers/
    - application_helper.rb
    - store_helper.rb
  - models/
    - category.rb
  - views/
    - layouts/
    - category/

Here starts the magic – scaffolding!

- app/controller/category_controller.rb
  - class CategoryController < ApplicationController
    scaffold :category
    end
- actions / methods / views for
  - new http://..../category/new
  - list http://..../category/list
  - show http://..../category/show/1
  - edit http://..../category/edit/1
  - destroy http://..../category/destroy/1

Adding validation to the model

- model/category.rb
  - class Category < ActiveRecord::Base
    validates_length_of :category, :within => 1..20
    validates_uniqueness_of :category,
    :message => "Categoria gia' presente"
    end

What's going on? Generative scaffolding

- ruby script/generate scaffold category
  - create app/views/categories/_form.rhtml
  - create app/views/categories/list.rhtml
  - create app/views/categories/show.rhtml
  - create app/views/categories/new.rhtml
  - create app/views/categories/edit.rhtml
  - create app/controllers/categories_controller.rb
  - create test/functional/categories_controller_test.rb
  - create app/helpers/categories_helper.rb
  - create app/views/layouts/categories.rhtml
  - create public/stylesheets/scaffold.css
- we've moved from the singular to the plural, so to use the new code you need to point your browser at http://localhost:3000/sampleapp/categories
  - delete the old category_controller.rb

class CategoriesController < ApplicationController

```
def index
  list
  render :action => 'list'
end

def list
  @category_pages, @categories = paginate :categories, :per_page
  => 10
end

def show
  @category = Category.find(params[:id])
end
```

...

```
def new
  @category = Category.new
end

def create
  @category = Category.new(params[:category])
  if @category.save
    flash[:notice] = 'Category was successfully created.'
    redirect_to :action => 'list'
  else
    render :action => 'new'
  end
end
```

. . .

```
def edit
  @category = Category.find(params[:id])
end

def update
  @category = Category.find(params[:id])
  if @category.update_attributes(params[:category])
    flash[:notice] = 'Category was successfully updated.'
    redirect_to :action => 'show', :id => @category
  else
    render :action => 'edit'
  end
end

def destroy
  Category.find(params[:id]).destroy
  redirect_to :action => 'list'
end
```

Notes

- CoC
  - action show
  - method show
  - render template ‘show.rthml’.
- render
  - render a different (non-conventional) template
- redirect_to
  - a different (non-conventional) page

Notes

- can use ActiveRecord methods such as find, find_all, new, save, update_attributes, destroy
- edit action
  - retrieve data from DB
  - render edit view
- update action
  - save data to DB
  - redirect to show

Software and Tutorials

- Software
  - www.rubyonrails.org
  - www.rubyforge.org
- Tutorial
  - “Four days on Rails” at rails.homelinux.org
- Book
  - “Agile Web Development with Rails” by Dave Thomas

Free book

- Sitepoint is currently *giving away* free PDF copies of Patrick Lenz's "Build Your Own Ruby on Rails Web Applications" book
  - http://www.sitepoint.com/books/rails1/freebook.php

- One controller per table
  - categories_controller.rb
- Multiple predefined actions per table
  - list, new, edit, show, destroy
- Multiple methods
  - list, new, edit, show, destroy + add, index, new, create, update

- http://localhost:3000/categories/list
  - Call list method of CategoriesController
- http://localhost:3000/categories/show/1
  - Call show method of CategoriesController with argument “1”

The (Action)View

- ActionView is the Rails class for the view
- Layouts
  - common view code, per controller, directory app/layouts
- Templates
  - action specific, per action, directory app/views/controllername
- Partials
  - small, reusable views

Templates

- processed by the render() method
- configuration option template_root
  - default: app/views
- Views for categories controller
  - app/views/categories

Rendering templates

- render(:action=> ‘myaction')
  - renders the template for action “myaction” in current controller
- render(:template=> ‘categories/list')
  - renders the template for action “list” in controller “categories”
- render(:file=> ‘/mydir/mytemplate.html')
  - renders the specified file (can be outside template_root)

Data visible to the template

- instance variables of the controller
- Implicit objects used by the controller
  - request, params,…
- current controller object
  - attribute “controller”
  - call class methods

Implicit objects

- **request:** client request. Contains several attributes
  - domain
  - remote_ip
  - env: environment of client browser
  - method, HTTP method (:get, :post, :delete, :put, :head)
  - delete?, get?, head?, post?, put? are methods which return true iff the request is of that type
- **params**
  - hash (associative array) containing parameters and values of the request.

Implicit objects

- **cookies**
  - associative array of cookies information
  - modifications are sent back
- **response**
  - server response, typically not used
- **session**
  - associative array of current user session information
- **headers**
  - associative array of response headers

Implicit objects

- Example
- if request.post?
  puts params[:category]
  cookie[:lastvisit] = Time.now

Templates

- .rxml templates
  - XML documents using Builder library
    - xml.category(“Computer Science”)
    - <category> Computer Science </category>
- .rhtml templates
  - HTML + embedded Ruby
    - <%=Time.now%>
    - Wed Oct 10 17:12:11 +0200 2007

Escaping HTML characters

- Direct output of user-submitted data is really bad habit: fragile, harmful, insecure
  - <%=params[:category]%>
- Solution: escape HTML char
  - html_escape() OR h()
  - <%=h params[:category]%>

- in views/categories/list.rhtml, line 13
  - <td><%=h category.send(column.name) %></td>
- If we write </body></html> as category name all is ok: chars are escaped
- If we remove the “h”, category name is blank (filtered out by rails)

Sanitize input

- sanitize() method
  - converts <form> and <script> tags into regular text,
  - remove all "onxxx" attributes (so that arbitrary Javascript cannot be executed).
  - removes href= and src= attributes that start with "javascript:“
- in ActionView::Helpers::TextHelper

Helpers

- Helper = module with methods for the view
- <%=truncate(@category.capitalize,10)%>
  - acceptable to put some code in templates
  - but not too much!
- one helper module per controller
  - CategoriesController -> CategoriesHelper

```
module CategoriesHelper
  def format_string
    …
  end
end
```

- <%=distance_of_time_in_words(Time.now,Time.local(2007,10,10))%>
  - 24 days
- <%=distance_of_time_in_words(Time.now,Time.now+33,false)%>
  - 1 minute
- <%=distance_of_time_in_words(Time.now,Time.now+33,true)%>
  - half a minute

- <%=time_ago_in_words(Time.local(2004,12,25)) %>
  - 116 days
- <%=number_with_precision(50.0/3)%>
  - 16.667
- <%=number_with_precision(50.0/3,1)%>
  - 16.7
- ……

- <%=simple_format(@text)%>
  - sort of text2html: line breaks, paragraphs
- <%=highlight(@text,”RoR”)%>
  - …RoR…
- <%=link_to 'Back', :action => 'list' %>
- <%=image_tag("/flower.png",
  :size=> “180x120")

- <%=mail_to(“a@a.it", "Support",

:subject=> “Hello I am #{@user.name}",

:encode=> "javascript")%>
- <%=stylesheet_link_tag “mystyle",
  :media=> "all" %>

Layout

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
  <title>Categories: <%= controller.action_name %></title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>
<p style="color: green"><%= flash[:notice] %></p>
<%= @content_for_layout %> <%#= yield  %>
</body>
</html>
```

- stylesheet_link_tag 'scaffold'
  - <link href="/stylesheets/scaffold.css" media="screen" rel="Stylesheet" type="text/css" />
- @content_for_layout
  - dynamic content, action-dependent

app/views/categories/new.rhtml

```
<h1>New category</h1>

<% form_tag :action => 'create' do %>
  <%= render :partial => 'form' %>
  <%= submit_tag "Create" %>
<% end %>

<%= link_to 'Back', :action => 'list' %>
```

- form_tag :action => 'create'
  - <form action="/categories/create" method="post"> … </form>
- submit_tag "Create"
  - <input name="commit" type="submit" value="Create" />
- link_to 'Back', :action => 'list'
  - Back

Partials

- views/categories/_form.html
- used in new.rhtml and edit.rhtml templates

```
<%= error_messages_for 'category' %>
<p><label for="category_category">Category</label><br/>
<%= text_field 'category', 'category‘ %></p>
<p><label for="category_created_on">Created on</label><br/>
<%= datetime_select 'category', 'created_on'  %></p>
<p><label for="category_updated_on">Updated on</label><br/>
<%= datetime_select 'category', 'updated_on'  %></p>
```

- error_messages_for
  - Returns a string with a DIV containing all of the error messages. This DIV can be tailored by the following options:
    - header_tag - Used for the header of the error div (default: h2)
    - id - The id of the error div (default: errorExplanation)
    - class - The class of the error div (default: errorExplanation)
    - object_name - The object name to use in the header
-
  <h2>1 error prohibited this category from being saved</h2>
  There were problems with the following fields:
  Category is too short (minimum is 1 characters)

- text_field 'category', 'category‘
  - <input id="category_category" name="category[category]" type="text" value="" />
- datetime_select
  - selects for Y, M, D, H, M,S

categories/new

- <!DOCTYPE …"> <head> …</head> <body>
- <h1>New category</h1> <form action="/categories/create" method="post">
- <label for="category_category">Category</label>
 <input id="category_category" maxlength="20" name="category[category]" size="20" type="text" value="" />
- <input name="commit" type="submit" value="Create" /> </form> Back
- </body> </html>

  - **LAYOUT**
  - **TEMPLATE**
  - **PARTIAL**

app/views/categories/list.rhtml

- <h1>Listing categories</h1>

```
<table> <tr>
  <% for col in Category.content_columns%>
    <th><%= col.human_name %></th>
  <% end %></tr>
```

- Category.content_columns
  - array of columns except “id”, “*_id”, “*_count”, and cols for single table inheritance
- col.human_name
  - nome_composto → Nome composto

```
<% for category in @categories %>
  <tr> <% for column in Category.content_columns %>
    <td><%=h category.send(column.name) %> </td>
  <% end %>
    <td><%= link_to 'Show', :action => 'show', :id => category %></td>
    <td><%= link_to 'Edit', :action => 'edit', :id => category %></td>
    <td><%= link_to 'Destroy', { :action => 'destroy', :id => category },
     :confirm => 'Are you sure?', :method => :post %></td> </tr>
<% end %></table>
```

- in categories_controller.rb

```
def list
  @category_pages, @categories = paginate :categories,
  :per_page => 10
 end
```

- :confirm => 'Are you sure?'
  - creates javascript confirmation dialog
- Pagination links
  - <%= link_to 'Previous page', { :page => **@category_pages**.current.previous } **if** **@category_pages**.current.previous %>
  - Previous page

Type mapping

- ActiveRecord automatically discovers the type of columns
  - char, varchar → String
  - int → Fixnum
  - decimal → Float
    - problem: decimal is precise, Float is not

Type mapping

SQL Type	Ruby Type
int, integer	Fixnum
decimal, numeric	Float
interval, date	Date
clob, blob, text	String
float, double	Float
char, varchar, string	String
datetime, time	Time
boolean	special handling needed

Accessing Attributes

- Two styles
  - account[:balance]
  - account.balance (preferred)
- Value is casted to a Ruby type
  - account.balance
    - returns a Float
  - account.balance_before_type_cast
    - returns a string

Booleans

- true/false, ‘t’/’f’, 1/0
- In Ruby
  - false iff **nil** OR the constant **false**
  - true otherwise (including, 0, ‘f’)

- DON’T DO THIS

```
if user.superuser
   grant_privileges
end
```

- DO THIS INSTEAD

```
if user.superuser?
    grant_privileges
end
```

- user.superuser?
  - false iff (0, ‘0’,’f’, ‘false’, ‘’, **nil**, **false**)
  - true otherwise
- Can be customized

```
class User<ActiveRecord::Base
        def superuser?
            self.superuser== ‘V'
        end
#...
end
```

Primary Keys and IDs

- convention: the primary key is named id
  - automatic management of values for new rows
- To override **set_primary_key “isbn”**
- Values for new rows must be managed assigning a value to objname.id
  - Notice: id is not an attribute, used only for setting primary key value
  - Ex: b = Book.new; b.id = “123982183223”

Creating new rows

- Create an object in memory
  - c = Category.new
- Set attribute values
  - c.category = “My brand new category”
- Save the object to the DB
  - c.save

- Without creating a local variable

```
Category.new do |c|
    c.category = “My brand new category”
    c.save
end
```

- A different constructor

```
c = Category.new(
  :category =>“My brand new category”
)
c.save
```

- puts "This category’s ID is #{c.id}"

- new() creates an object in memory
- save() stores it to the database
- create() creates the object and stores it to the DB

```
c = Category.create(
  :category =>“My brand new category”
)
c = Category.create(
  [
    {:category =>“My brand new category”},
    {:category =>“My second brand new category”}
  ]
)
```

Read rows

- find() finds records using primary key(s)
- find(12)
  - returns the corresponding object or the RecordNotFound exception
- find([12,45,78,90]) OR find(12,45,78,90)
  - returns the corresponding objects or the RecordNotFound exception if any of the ids cannot be found

- Category.find(:all,
  :conditions => “created_on < ‘2010-10-10’”)
- Options
  - :first returns only the first matching row
  - :all returns all matching rows
- DON’T DO THIS (SQL Injection attack)

  date = params[:max_date]

  Category.find(:all,
  :conditions => “created_on < #{date}”)

SQL Injection in one slide

- Code
  - SELECT *fieldlist* FROM *table*
    WHERE *field* = '$EMAIL';
- Legitimate
  - SELECT *fieldlist* FROM *table*
    WHERE *field* = 'steve@unixwiz.net';
- Attack
  - SELECT *fieldlist* FROM *table*
    WHERE *field* = 'anything' OR 'x'='x';
- References
  - http://www.unixwiz.net/techtips/sql-injection.html
  - http://en.wikipedia.org/wiki/SQL_injection

- DO THIS INSTEAD

  ```
  date = params[:max_date]
  Category.find(:all,
  :conditions => [“created_on < ?”, date])
  ```
- If using multiple input parameters
-
  ```
  date1 = params[:min_date]
  date2 = params[:max_date]
  Category.find(:all,
  :conditions =>
    [“created_on > :mind AND created_on < :maxd”,
          {:mind => date1, :maxd => date2}
    ])
  ```

- Category.find(:first)
  - which record is returned?
  - SELECT * FROM categories LIMIT 1
- Category.find(:all,

  :select => “”
  :conditions => “……”
  :order => “category, created_on DESC”
  :limit => “20”
  :offset => “10”
  :joins => “…sql code for joining other tables…”

- find_by_sql(‘SELECT * FROM categories WHERE (created_on < '2010-10-10')’)
- Querying attributes
  - attributes(): returns associative array
  - attribute_names()
  - attribute_present?(“category”)

Counting rows

- Category.count
  - SELECT count(*) AS count_all FROM categories
- Category.count([“created_on < ?”, date])
- Category.count_by_sql(“SELECT * FROM categories WHERE (created_on < '2010-10-10')”)

Dynamic finders

- find_by (uses :first)
- find_all_by (uses :all)
- Examples
  - Category.find_all_by_category(‘Network’)
  - Category.find_by_category(‘My category’)
  - find_by_category_and_date(cat,date)
    - equivalent to Category.find(:first, ["category = ? AND date = ?", cat, date])
  - there is NO find_by… _or_...

Reloading Data

- Multiple concurrent processes accessing the DB
  - Objects in memory may differ by rows in DB
- reload()
  - reloads data from DB
  - used in unit test

Updating Existing Rows

- save()
  - creates a new row, or
  - updates an existing row
- updates all the columns corresponding to attributes
- c = Category.find(34); c.category=“aa”; c.save;
  - updates all the columns
- c = Category.find_by_sql(“SELECT id, category FROM categories”); c.category=“aa”; c.save;
  - only updates columns id and category

- update_attribute() updates object attributes and saves changes to the DB
- c = Category.find(34)
  c. update_attribute(:category=>“aa”)
- c = Category.find(34)
  c. update_attributes(:id => 1034
  :category=>“aa”)

- update() reads, updates and saves to DB
  - class method
- Category.update(34,:category=> “aaa")
- Category.update_all(sql_set, sql_where)

- save
  - if something goes wrong simply returns false
- save!
  - if something goes wrong raises a RecordNotSaved exception

Race condition

- Multiple concurrent processes
  - DB contains row
    - id=23, category="New Category", description = ""
  - P1: c=Category.find(23)
  - P2: c=Category.find(23)
  - P1: c.category="Ruby"
  - P2: c.description="Ruby on Rails"
  - P1: c.save
    - id=23, category="Ruby", description = ""
  - P2: c.save
    - id=23, category="New Category", description = "Ruby on Rails"

Pessimistic locking

- Try to acquire lock on row
- If row is locked wait and retry later
- If row is not locked acquire the lock
- Work on the row
- Release the lock

- Problems
  - low concurrency
  - processes may not release locks

Optimistic Locking

- Does not acquire locks
- Just before writing data back to the DB verify that row is unchanged
  - Use a version number
- If not, throw an exception (ActiveRecord::StaleObjectError) → must be catched
- Rails automatically enables optimistic locking if the table contains a column named *lock_version* type integer (SQL default=0)
- unless *ActiveRecord::Base.lock_optimistically= false*

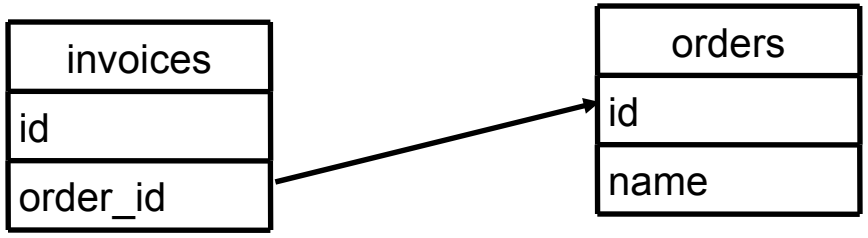
Deleting Rows

- delete(12) or delete([12,34,54])
- delete_all(conditions = nil)
  - Deletes all the records that match the condition without instantiating the objects first (and hence not calling the destroy method).
  - Example: Post.delete_all "person_id = 5 AND (category = 'Something' OR category = 'Else')“
- delete/delete_all bypass ActiveRecord
  - more efficient
- Use destroy / destroy_all if you want to mantain consistency according to the model rules (executes all associated callbacks)

Table relationships

- One-to-one
  - has_one in the referenced table
  - belongs_to in the referencing table (which contains the foreign key)
- One-to-many
  - has_many in the referenced table
  - belongs_to in the referencing table
- Many-to-many
  - has_and_belongs_to_many in both tables

One-to-one




```
class Invoice < ActiveRecord::Base
  belongs_to  :order
end
```

```
class Order < ActiveRecord::Base
  has_one  :invoice
end
```

One-to-one

- One-to-one is One-to-zero-or-one
- Relation can be established in two ways
  - Case 1: from the referenced model object
  - Case 2: from the referencing model object
- By convention
  - the foreign key column is named <tablename>_id
    - ex: order_id
  - the class name is derived from <tablename> as usual

- Case 1
  - my_invoice=Invoice.new(...)
  - my_order.invoice=my_invoice

- NOTICE: object my_invoice is saved to the DB
- NOTICE: any previously associated invoice object is updated: its foreign key values is set to NULL

- Case 2
  - my_order=Order.new(...)
  - my_invoice.order=my_order #Order will not be saved
- NOTICE: object my_invoice is NOT saved to the DB

- There can be multiple associations

```
class LineItem<ActiveRecord::Base
    belongs_to: product
    belongs_to: invoice_item
end
```

- Conventions can be overridden

```
class LineItem<ActiveRecord::Base
    belongs_to :paid_order,
        :class_name=> "Order",
        :foreign_key=> "order_id",
        :conditions=> "paid_on is not null"
end
```

Added methods belongs_to

```
class LineItem<ActiveRecord::Base
    belongs_to :product
end
```

- LineItem gets the following instance methods
- product(force_reload=false)
  - result is cached
- product=(obj)
  - associate a product to lineitem. Product saved when lineitem is saved
- build_product(attributes={})
  - create and associate a product to lineitem. Product saved when lineitem is saved.
- create_product(attributes={})
  - like build_product but saves the product
- nil? - returns true if there is no associated object.

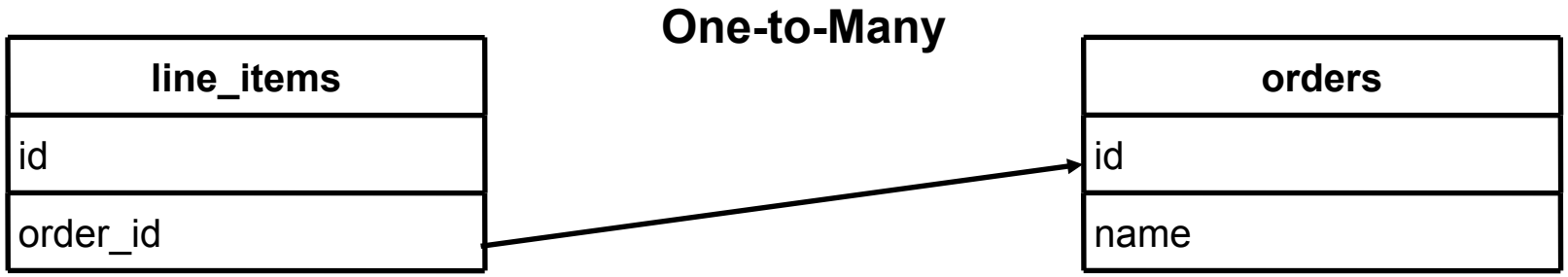
Added methods :has_one

- Same methods as belongs_to
- Same options
  - :class_name, :foreign_key, :conditions
- Is it a belongs_to or has_one association?
  - both express a 1-1 relationship
  - the foreign key goes on the table for the class saying belongs_to
  - there is only one base class (saying has_one)

has_one… :dependent

- :dependent=>:destroy (or true)
  - delete the associated row in the child table
- :dependent=>:nullify
  - child row is orphaned by setting FK to null
- :dependent=>false (or nil)
  - child row not updated

One-to-many




```
class LineItem < ActiveRecord::Base
      belongs_to  :order
end
```

```
class Order < ActiveRecord::Base
      has_many  :line_items
end
```

has_many

- has_many defines an attribute that behaves like a collection of child objects.
  - Access children in an array
  - Find, add, remove, update children

```
order=Order.new
qty = 2
product=Product.find(123)
order.line_items<<LineItem.new(
    :product=>product
    :quantity=>qty)
```

has_many

- << is the append operator
  - adds the object to the list
  - sets the value of the foreign key product_id in the lineitem
  - saves the lineitems when the order is saved
- Iterating over line items

```
order=Order.find(123)
total=0.0
order.line_items.each do |li|
    total+=li.quantity * li.unit_price
end
```

- Same options
  - :class_name, :foreign_key, :conditions, :dependent
- :exclusively_dependent
  - child rows deleted with a single SQL query (if child table used only by parent table)
- :order
  - use a particular order
  - Ex: :order=> "quantity,unit_price DESC"

has_many

- line_items.build(attributes={}) line_items.create(attributes={})
- line_items.push(object, …)
- line_items.delete(object, …)
  - removes one or more objects from the collection
  - set FK to NULL or destroy if :dependent
- line_items.clear
  - removes every object from the collection
- line_items_singular_ids line_items.uniq
  - array of the associated objects ids
- line_items.empty?
  - true if there are no associated objects
- line_items.size (cached results) OR .length (fresh results)
- line_items.find(…)

books

id	title	author	date	publisher_id
1	Agile Web Development with Rails	Thomas, Dave	2005-11-17	1
2	Programming Ruby	Thomas, Dave	2005-01-02	1
3	Web Component Development with Zope 3	Weitershausen	2005-05-01	2

publishers

id	name
1	The Pragmatic Bookshelf
2	Springer

```
CREATE TABLE `books` (
  `id` int(11) NOT NULL auto_increment,
  `title` varchar(100) NOT NULL,
  `author` varchar(50) NOT NULL,
  `date` date default '0000-00-00',
  `publisher_id` int(11) NOT NULL,
  PRIMARY KEY  (`id`)
)
constraint fk_publisher       foreign key (publisher_id) references publishers(id)
```

```
CREATE TABLE `publishers` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(50) NOT NULL,
  PRIMARY KEY  (`id`)
)
```

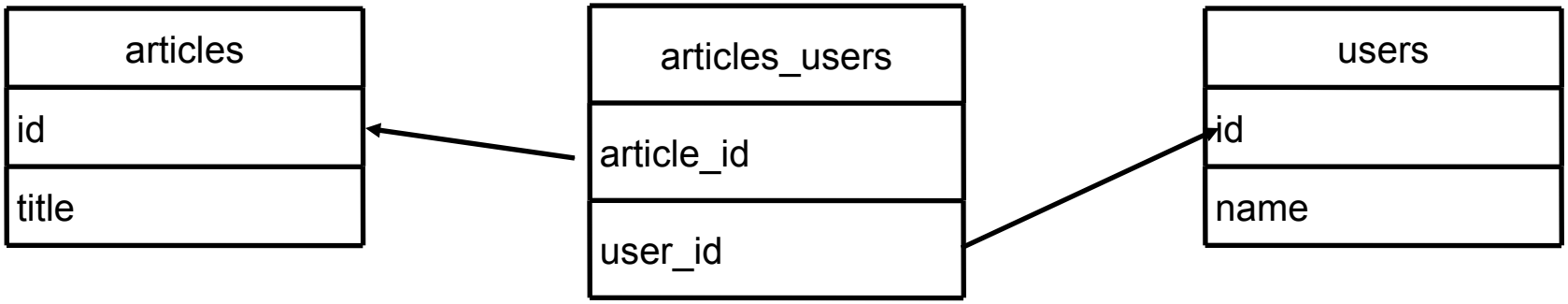
```
class Book < ActiveRecord::Base
  belongs_to :publisher
end

class Publisher < ActiveRecord::Base
  has_many :books
end
```

```
pragprog = Publisher.create(:name => ‘Prag Prog')

awdr = Book.new
awdr.title = ‘Agile Web Development with Rails'
awdr.author = ‘Dave Thomas'
awdr.date = Time.now
awdr.publisher = pragprog
awdr.save

book = Book.find_by_author('Dave Thomas')
puts book.publisher.name
```

Many-to-Many

- Associates two classes via an intermediate join table.
- join table guessed composing class names using lexic. order
  - join btw Developer and Project default join table name of "developers_projects"
  - but can be explicitly specified

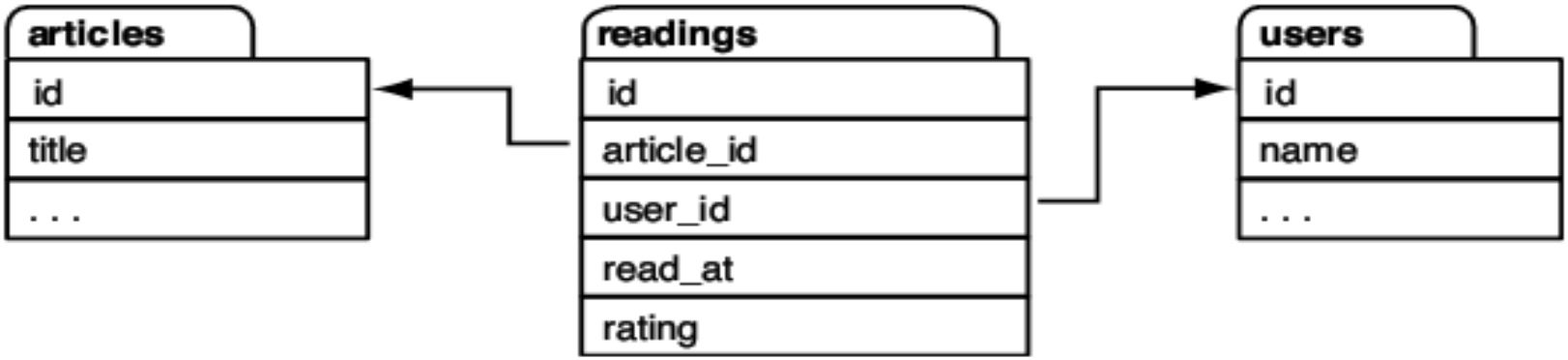
Many-to-Many




```
class Article < ActiveRecord::Base
    has_and_belongs_to_many :users
end
```

```
class User < ActiveRecord::Base
    has_and_belongs_to_many :articles
end
```

```
#Who has read article 123?
article=Article.find(123)
readers=article.users

#What has Dave read?
dave=User.find_by_name("pragdave")
articles_that_dave_read=dave.articles

#How many times has each user read article123
counts=Article.find(123).users.count(:group=> "users.name")
```




```
class Article < ActiveRecord::Base
  has_many :readings
end

class User < ActiveRecord::Base
  has_many :readings
end

class Reading < ActiveRecord::Base
  belongs_to :article
  belongs_to :user
end
```

```
reading = Reading.new
reading.rating  = params[:rating]
reading.read_at = Time.now
reading.article = current_article
reading.user    = session[:user]
reading.save
```

- We lost the possibility to navigate between models using article.users or user.articles
- Solution : use the :through option

```
class Article < ActiveRecord::Base
  has_many :readings
  has_many :users, :through => :readings
end

class User < ActiveRecord::Base
  has_many :readings
  has_many :articles, :through => :readings
end
```

```
  has_many :readers, :through => :readings, :source => :user
  has_many :happy_users, :through => :readings,  :source => :user,
           :conditions => 'readings.rating >= 4'
```

Removing duplicates

```
class Article < ActiveRecord::Base
  has_many :readings
  has_many :users, :through => :readings, :uniq => true
end
```