Rolling with Ruby on Rails

Автор: Curt Hibbs
Перевод: Андрей Горбатов
Оригинал статьи

Возможно, вы слышали о Ruby on Rails, новом эффективном способе разработки веб-приложений, и хотите его попробовать, но ничего о нем не знаете. В этой статье разрабатывается приложение с использованием Rails. Здесь вы не научитесь программировать на Ruby, но если вы знакомы с другим объектно-ориентированным языком, вам не составит труда прочесть этот туториал до конца (и в конце вы найдете ссылки на ресурсы по изучению Ruby).

Давайте ответим на пару вопросов перед началом!

Что такое Ruby?

Ruby - это полностью объектно-ориентированный язык программирования с прозрачным синтаксисом, что делает программированием на нем изящным и веселым. Ruby с успехом совмещает изящность языка Smalltalk, простоту использования и изучения Python и практичность Perl. Ruby родился в Японии в 1990 годах и стал популярен по всему миру за последние несколько лет в связи с появлением англоязычной литературы по нему.

Что такое Rails?

Rails – это open source Ruby фреймворк для разработки веб-приложений, имеющих базу данных. Что же в этом особенного? Ведь есть много других фреймворков, которые появились намного раньше Rails. Почему мы должны выбирать этот?

Что вы подумаете, если я вам скажу, что вы сможете разработать веб-приложение с использованием Rails по крайней мере в 10 раз быстрее, чем используя типичный Java фреймворк? А вы сможете – даже без жертв в качестве приложения! Как же это возможно?

Первая часть ответа – это язык программирования Ruby. Многие простые вещи, которые можно сделать на Ruby, недоступны в других языках программирования. Rails пользуется этим на 100%. Оставшаяся часть ответа – это два принципа Rails: *меньше кода* и *соглашение вместо конфигурации*.

Меньше кода значит, что вам приходится писать меньше строк кода, чтобы реализовать своё приложение. А чем меньше кода, тем быстрее идёт разработка и тем меньше багов, что делает код легким для понимания, поддержки и расширения. Очень скоро вы увидите, как Rails скинет с вас огромный груз кода. (Абзац взят статьи интернет-журнала Дмитрия Сабанина).

Соглашение вместо конфигурации означает конец подробным файлам конфигурации на XML — в Rails нет ни одного! Вместо трудоёмкого XML, приложение на Rails использует несколько простых программерских соглашений, которые позволяют узнать всё через рефлекцию и обнаружение. Например, Rails использует разумную рефлекцию для привязки таблиц базы данных к объектам Ruby. Код вашего приложения и работающая база данных уже содержат всё, что Rails нужно было знать. (Абзац взят статьи интернет-журнала Дмитрия Сабанина).

Смотреть значит верить

Мы, разработчики, очень часто слышим радостные возгласы, сопровождающие что-либо новое. Я только могу представить ваш скептический взгляд, когда вы слышите мои сомнительные заявления. *В десять раз быстрее, конечно!*

Я не прошу слепо верить мне. Я все вам докажу! Но сперва нужно установить необходимое программное обеспечение. А затем перейдем непосредственно к разработке приложения.

Установка ПО

Мы будем разрабатывать приложение в среде Windows. Если у вас Linux или Macintosh, вы все равно можете прочесть эту статью, но скриншоты будет немного отличаться от приведенных здесь. И еще вам необходимо установить ПО, разработанное специально для вашей системы. Смотрите раздел Resources в конце данной главы.

Чтобы разработать наше приложение, необходимо установить следующее ПО:

- Ruby
- Rails фреймворк
- Базу данных MySQL (и GUI клиент MySQL-Front)

Шаг 1: Установка Ruby

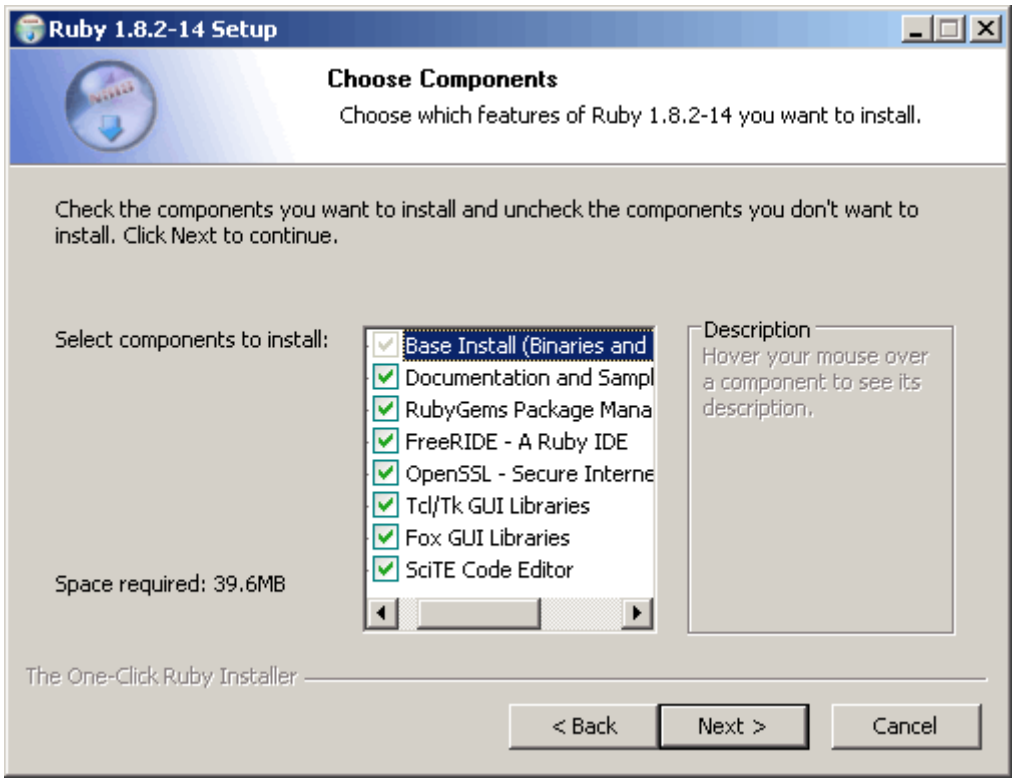

Рисунок 1. Окно установки Ruby в Windows

Установка Ruby очень проста:

1. Загрузите последнюю версию Ruby (Инсталлируем Ruby для Windows в один клик)
2. Запустите установочный файл и следуйте инструкциям. Для простоты жмите `Enter`, чтобы принять все настройки по умолчанию.

Замечание пользователям Linux и OS X: Инсталлятор для Windows идет вместе с пакетом RubyGems. Это значит, что если вы установили Ruby, вам необходимо установить еще RubyGems.

Шаг 2: Установка Rails

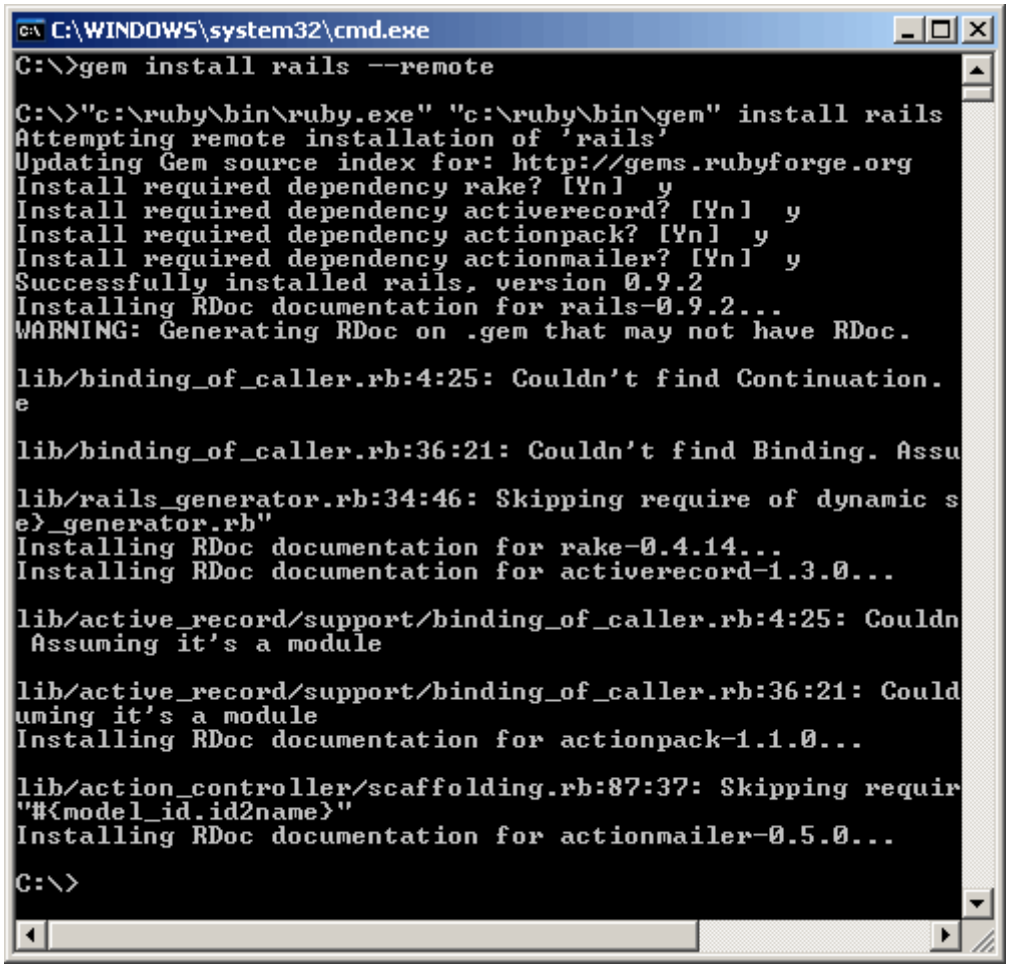

Рисунок 2. Установка Rails через RubyGems

Теперь мы можем использовать RubyGems, чтобы загрузить и установить Rails. Как показано на рисунке 2:

1. Откройте командную строку и запустите команду `gem install rails --remote`.
2. RubyGems также установит все библиотеки, от которых зависит Rails. Для каждой RubyGems будет запрашивать разрешение на установку. Отвечайте "y" (yes).

Шаг 3: Установка MySQL

Рисунок 3. Окно установки MySQL Server

Нам необходимо установить сервер базы данных. Rails поддерживает много типов баз данных. Мы будем использовать MySQL.

1. Загрузите последнюю "essential" версию MySQL.
2. Запустите файл установки и принимайте все настройки по умолчанию, но пропустите регистрацию для mysql.com.
3. После окончания установки нажмите кнопку Finish и вы попадете в раздел настроек.
4. Здесь вы также можете оставить все настройки по умолчанию, кроме чекбокса "Modify Security Settings" (Рисунок 4). Это из-за того, что начиная с версии 4.1.7, MySQL использует новый алгоритм аутентификации, что несовместимо со старым клиентским ПО, включая текущую версию Rails. Убрав галку с этого чекбокса, вы получаете беспарольный доступ к MySQL.

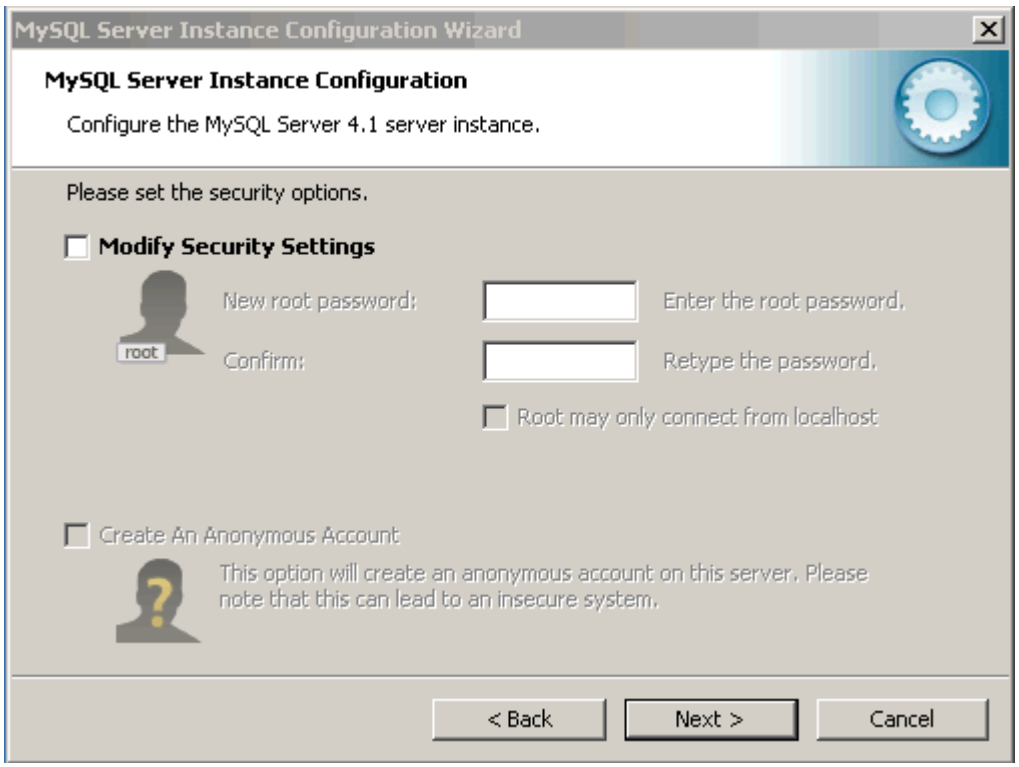

Рисунок 4. Окно конфигурации MySQL

Шаг 4: Установка MySQL-Front

MySQL-Front – это графический интерфейс для базы данных MySQL. Это недорогой коммерческий продукт с триальным периодом 30 дней. В нашей статье мы используем MySQL-Front для разработки нашей базы данных. Вы также можете воспользоваться командной строкой для работы с MySQL.

1. Загрузите последнюю версию MySQL-Front.
2. Запустите установочный файл и оставьте все настройки по умолчанию (Рисунок 5).

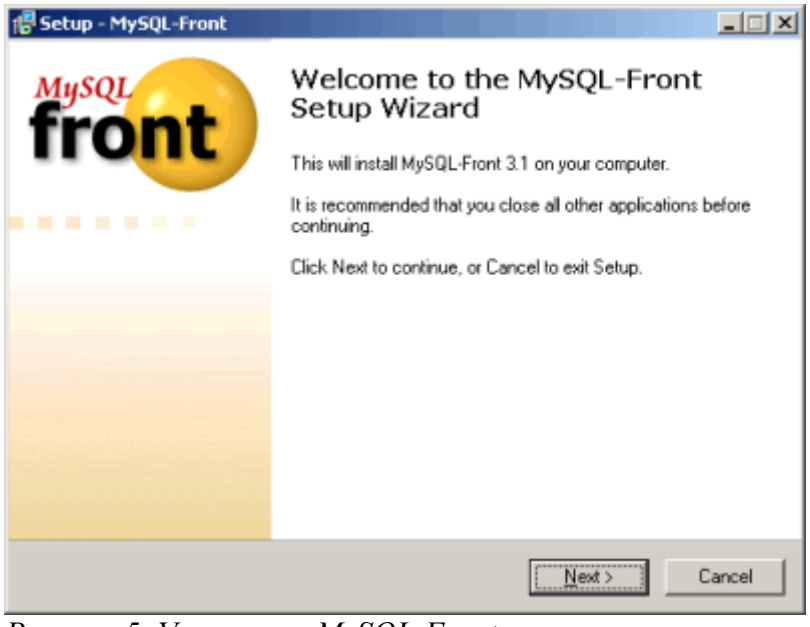

Рисунок 5. Установка MySQL-Front

Давайте писать код

Мы создадим онлайновую кулинарную книгу (**cookbook**) для хранения рецептов. Функционал книги следующий:

- Отображать все рецепты.
- Создавать новые и редактировать существующие рецепты.
- Назначать категории рецептам ("десерт" или "суп").

Вы можете создать приложение в любой папке, я использовал *c:\rails\cookbook*. Все команды вызываются отсюда.

Если хотите, то можете полный пример Rails cookbook приложения. Только не забудьте скорректировать в папке *cookbook/db/* файл настроек базы данных.

Создание пустого Rails приложения

Rails – это не только runtime фреймворк для веб-приложений, но и набор скриптов-помощников, автоматизирующих вашу работу. На этом этапе мы будем использовать скрипт-помощник, создающий структуру каталогов и начальный набор файлов приложения. Откройте командную строку, зайдите в каталог, где вы хотите разместить приложение (я использовал *c:\rails*.)

1. Запустите команду:

   ```
   rails cookbook
   ```

   Создастся каталог *cookbook*, содержащий полный набор папок и файлов для пустого приложения Rails application.

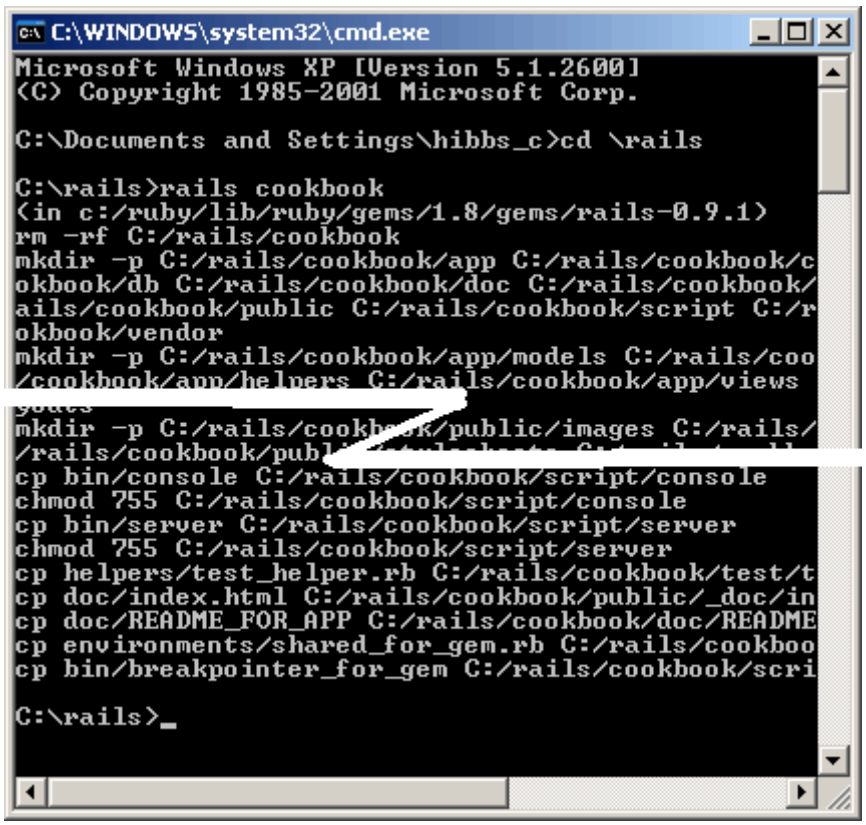

Рисунок 6. Созданные папки для приложения Rails

Тестирование пустого веб-приложения

Приложение Rails может быть запущено на любом виртуальном сервере, но более удобно использовать встроенный сервер WEBrick. Давайте запустим наш сервер

- Откройте командное окно в каталоге cookbook.
- Выполните команду:

  ```
  ruby script\server
  ```

  чтобы запустить сервер (Рисунок 7).

- Теперь откройте браузер и перейдите на http://127.0.0.1:3000/. Вы увидите что-то наподобие рисунка 8.

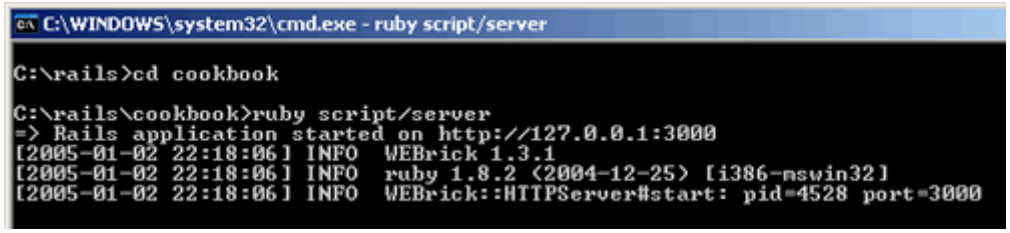

Рисунок 7. Запуск сервера WEBrick

Оставляйте командную строку открытой, чтобы работал сервер.

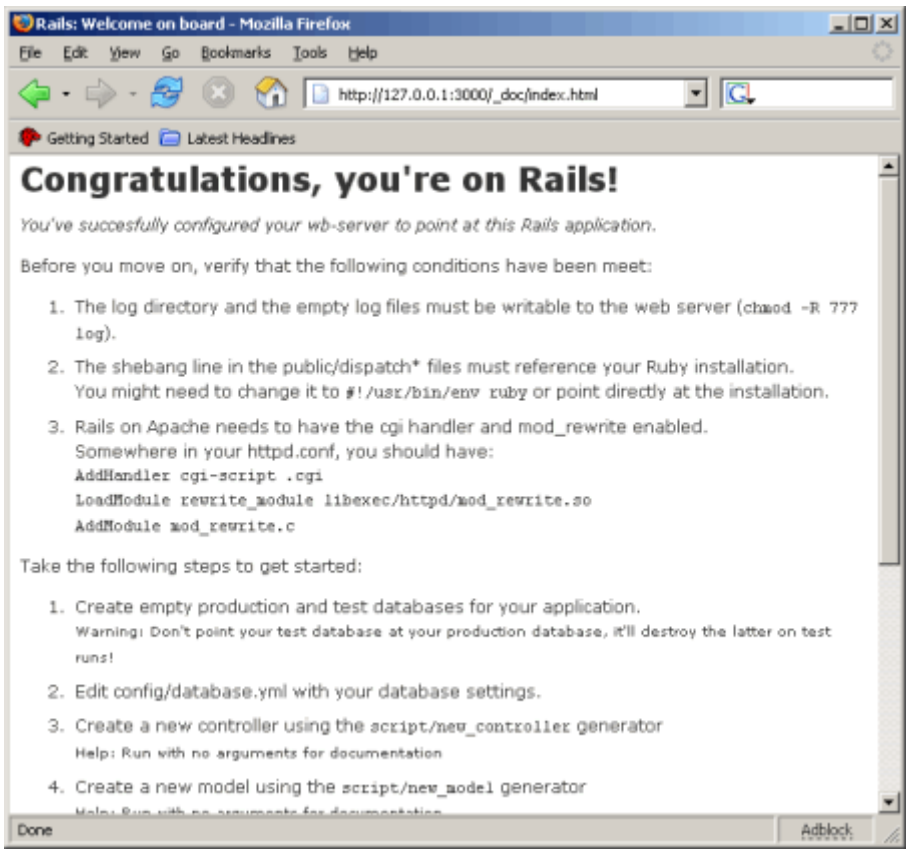

Рисунок 8. Страница по умолчанию Rails

Структура каталогов приложения Rails

Rails старается уменьшить количество лишней работы. Когда вы использовали скрипт-помощник, он создал полную структуру каталогов приложения (Рисунок 9). Rails знает, что и где найти в этой структуре, поэтому вам не надо ему об этом сообщать. *Запомните, никаких конфигурационных файлов!*

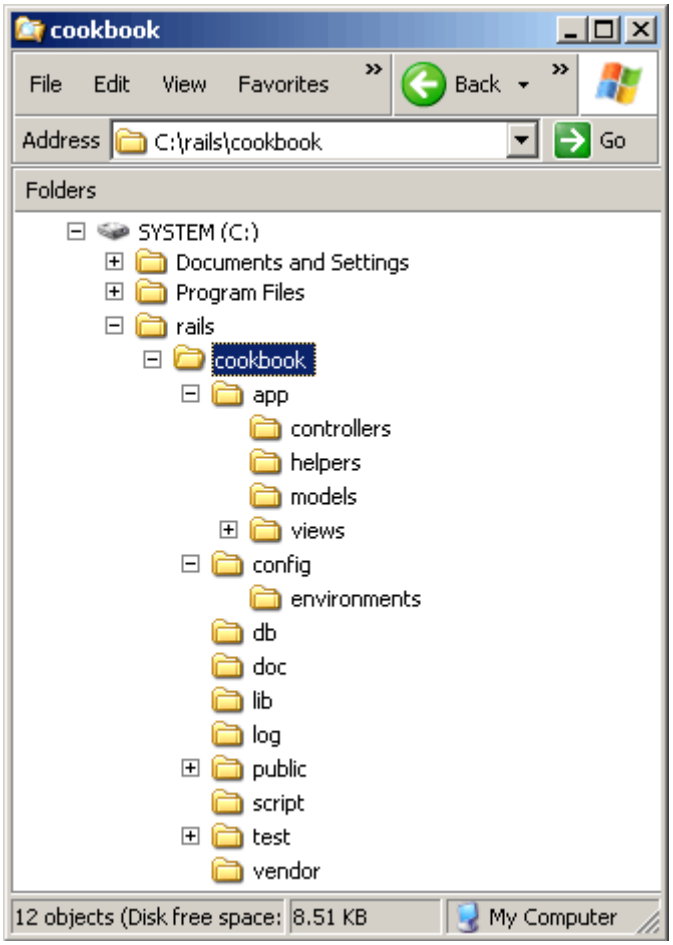

Рисунок 9. Структура каталогов приложенияRails

В основном процесс разработки состоит в создании и редактировании файлов в подкаталогах *c:\rails\cookbook\app*. Расскажем кратко об их использовании.

- Каталог *controllers* – место, где Rails ищет классы контроллеров. Контроллер обрабатывает запрос от пользователя.
- Каталог *views* содержит HTML шаблоны, которые заполняются данными и отображаются в браузере.
- Каталог *models* содержит классы, которые моделируют и преобразуют данные, хранящиеся в базе данных. Во многих фреймворках эта часть обычно огромна и подвержена ошибкам. Rails же делает это предельно просто!
- Каталог *helpers* содержит классы помощников, использующиеся с классами моделей, видов и контроллеров. Это позволяет хранить код классов в порядке.

Контроллеры и URL

Важно понять, как работают контроллеры в Rails и как отображаются URL в методах контроллеров.

Классы контроллеров обрабатывают запросы пользователя. URL запроса указывает контроллеру на метод внутри класса. Как это работает?

Оставьте командную строку с запущенным сервером открытой и откройте вторую. Зайдите в каталог *c:\rails\cookbook*. Он будет выглядеть как на рисунке10.

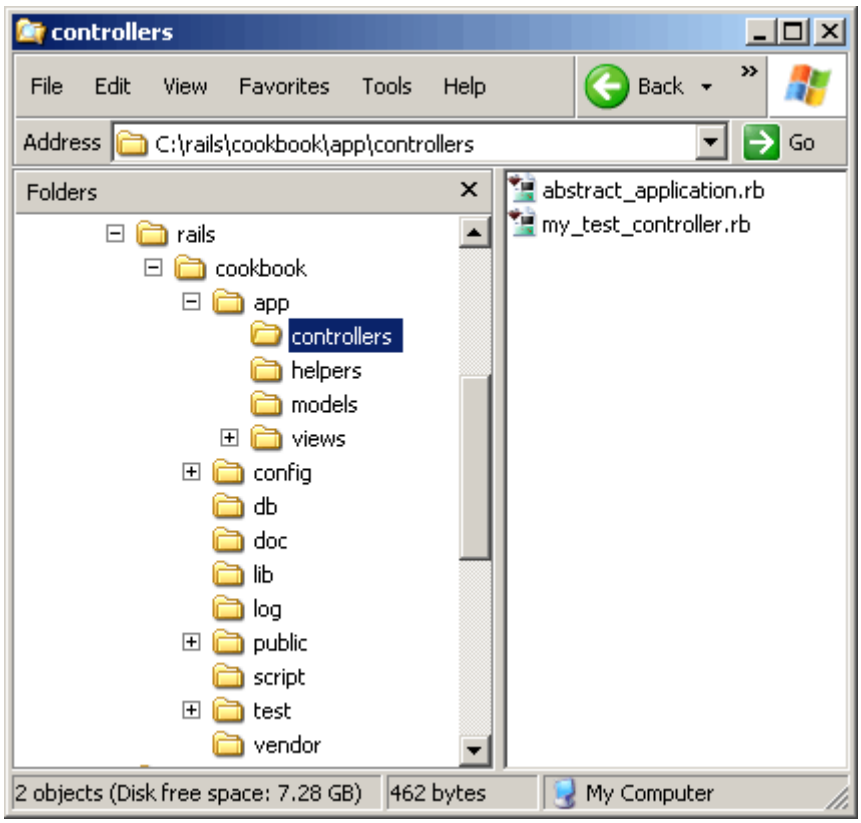

Рисунок 10. Каталог контроллер controller

Теперь используем другой скрипт-помощник для создания нового класса контроллера. Выполните команду:

```
ruby script\generate controller MyTest
```

Этот скрипт создаст файл с именем *my_test_controller.rb*, содержащий основу для класса `MyTestController`.

В каталоге *c:\rails\cookbook\controllers* откройте этот файл для редактирования:

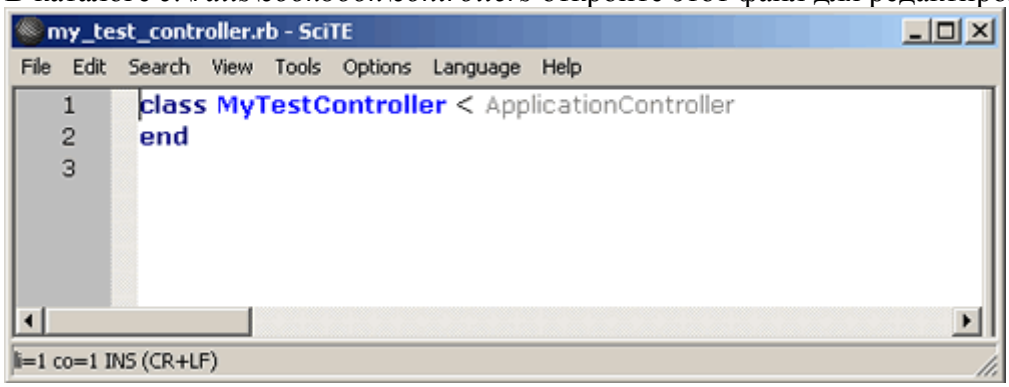

Рисунок 11. Редактирование `MyTestController`

Что произойдет, если вы перейдете на место, которого не существует? Попробуйте `http://127.0.0.1:3000/garbage/`. Результат показан на рисунке 12.

Рисунок 12. Переход на неизвестный контроллер

Ничего удивительного. Теперь попробуйте `http://127.0.0.1:3000/My Test/`, см. Рисунок 13.

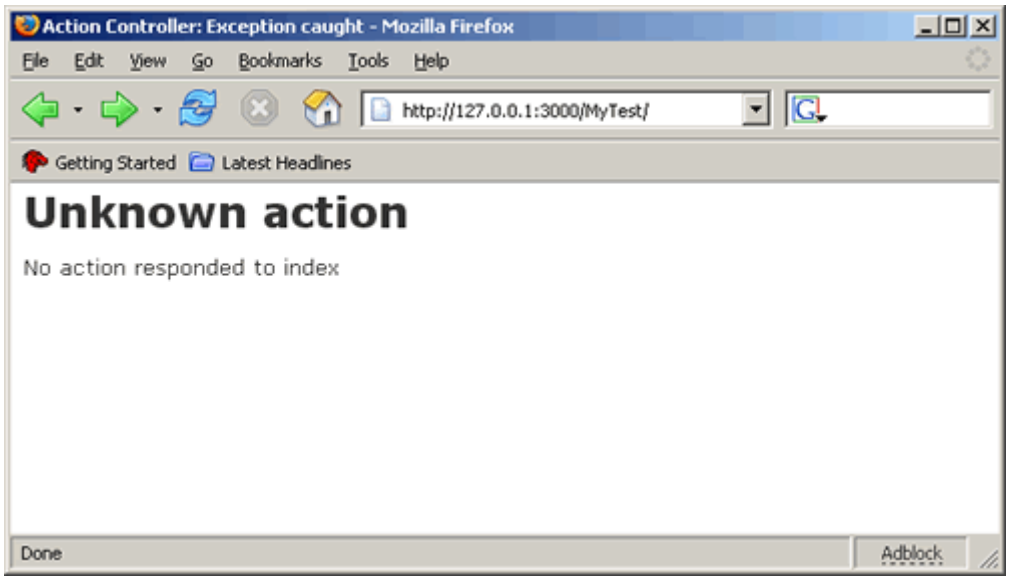

Рисунок 13. Переход на новый контроллер

Хм. Что-то другое. Часть ссылки `MyTest` указывает на новоиспеченный контроллер. Кажется Rails пытается и не может найти действие с именем `index` в этом контроллере.

Исправим это. Добавим метод `index` как показано на рисунке 14.

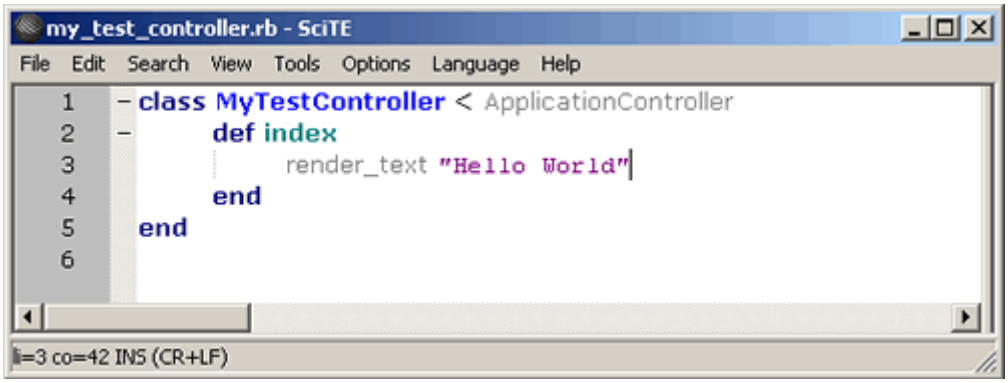

Рисунок 14. Метод `index` контроллера `MyTestController`

Обновите страницу и вы увидите то, что показано на рисунке 15.

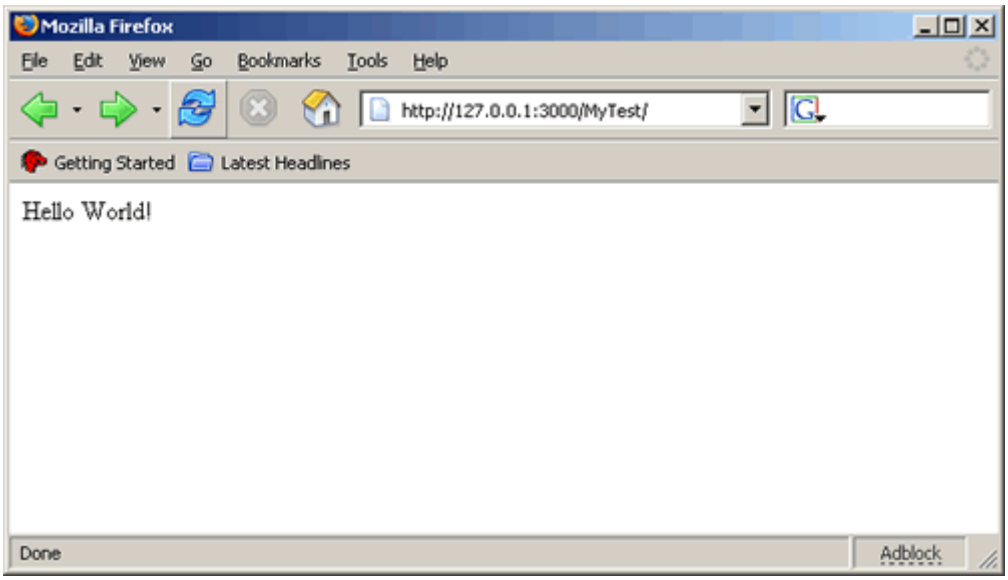

Рисунок 15. Результат метода `index`

Тех же результатов вы добьетесь, пройдя по ссылке `http://127.0.0.1:3000/My Test/index`.

Давайте добавим другое действие в контроллер чтобы окончательно убедиться. Добавьте метод `dilbert` как показано на рисунке 16.

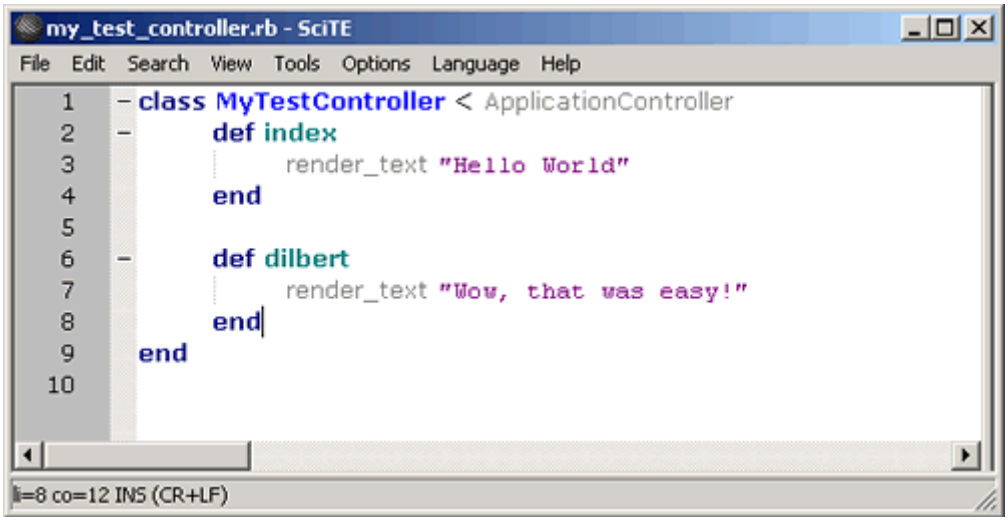

Рисунок 16. Метод `dilbert`

Перейдите на http://127.0.0.1:3000/My_Test/dilbert и увидите что-то похожее на Рисунок 17.

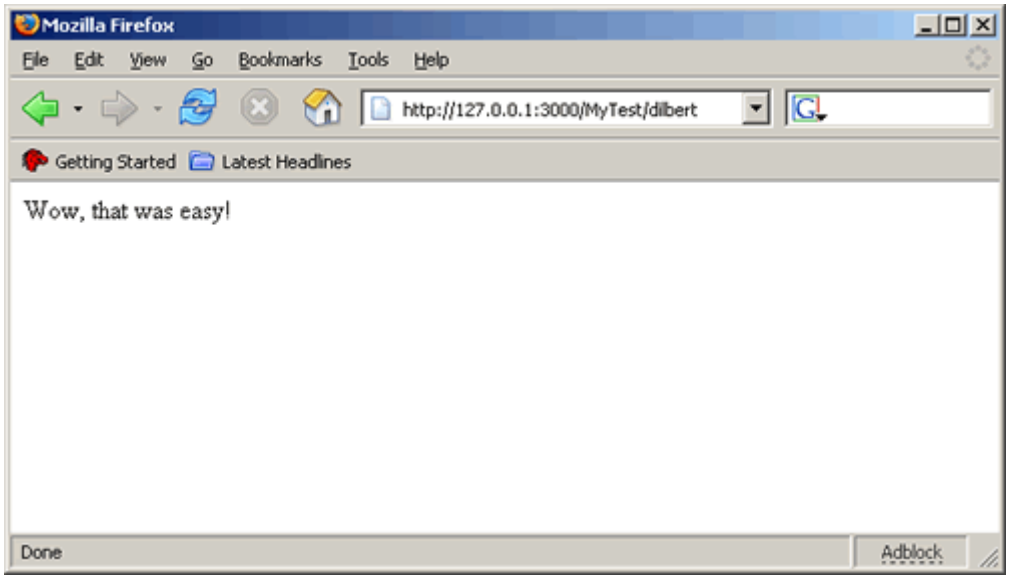

Рисунок 17. Вывод метода `dilbert`

Давайте создадим нашу базу данных.

Создание базы данных Cookbook

Настало время создать базу данных и сказать Rails, как ее найти. (Это единственная настройка, которую вы найдете в Rails.)

1. Запустите MySQL-Front и войдите в локально запущенный MySQL (`localhost`) как root, используя пустой пароль. Вы увидите что-то похожее на рисунок 18.

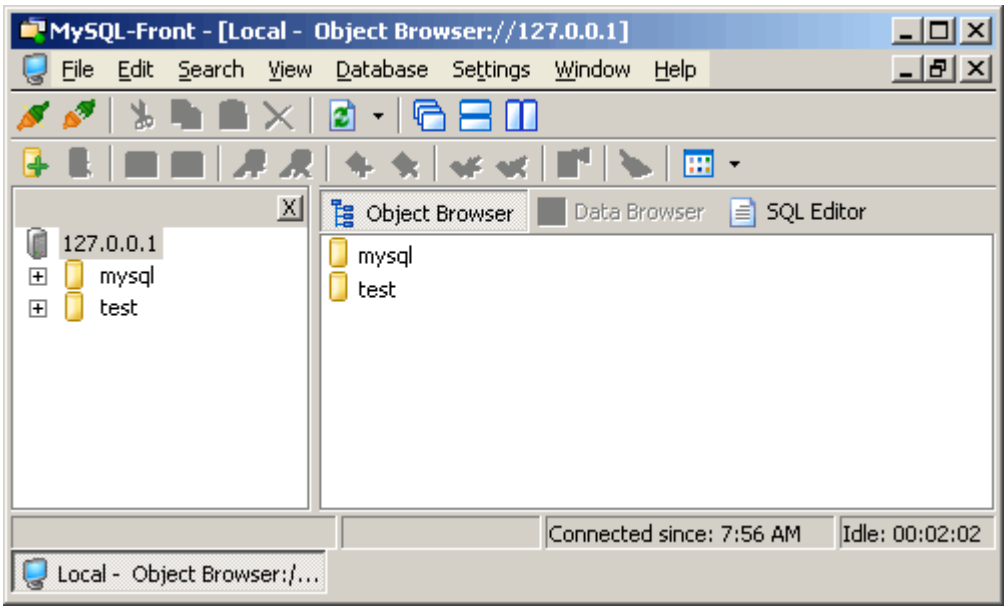

Рисунок 18. MySQL-Front

2. Сейчас есть две базы данных, `mysql` и `test`. Создайте новую базу данных и назовите ее `cookbook`. (Меню `Database>New>Database и введите имя cookbook`, как показано на рисунке 19).

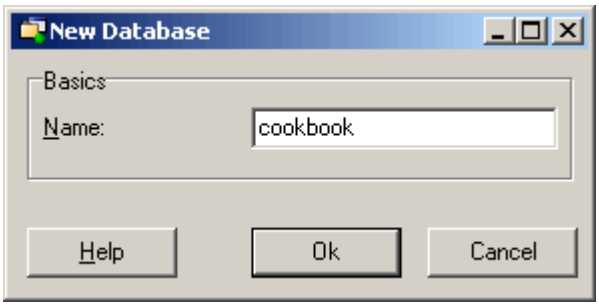

Рисунок 19. Создание новой базы данных

Нажмите Ok для создания базы данных.

3. Чтобы сообщить Rails, как найти базу данных, отредактируйте файл *c:\rails\cookbook\config\database.yml* и измените название базы данных на `cookbook`. Оставьте имя пользователя root и пароль пустым. Когда вы закончите должно выглядеть как на Рисунке 20.

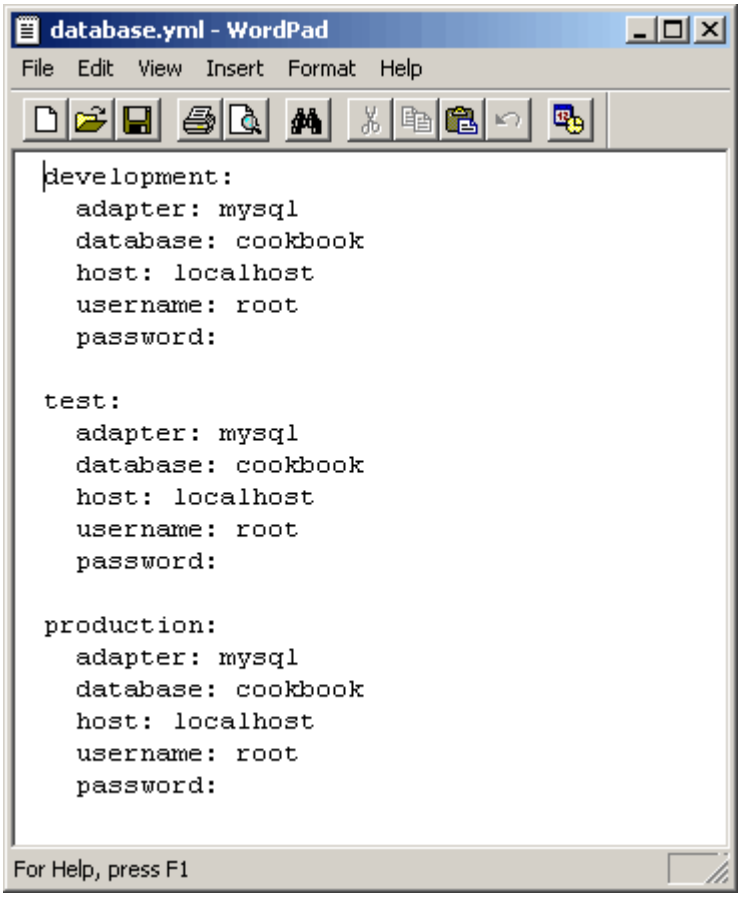

Рисунок 20. Файл конфигурации database.yml

Rails позволяет запускать приложение в режиме разработки, тестирования и производства, используя различные базы данных. Наше приложение использует одну базу данных.

Чтобы изменения этого файла вступили в силу, необходимо перезапустить сервер.

Создание таблицы `recipes`

Наша кулинарная книга будет содержать рецепты, поэтому давайте создадим таблицу для их хранения.

В левой панели MySQL-Front, кликните правой кнопкой по базе данных cookbook и выберите `New>Table` (Рисунок 21).

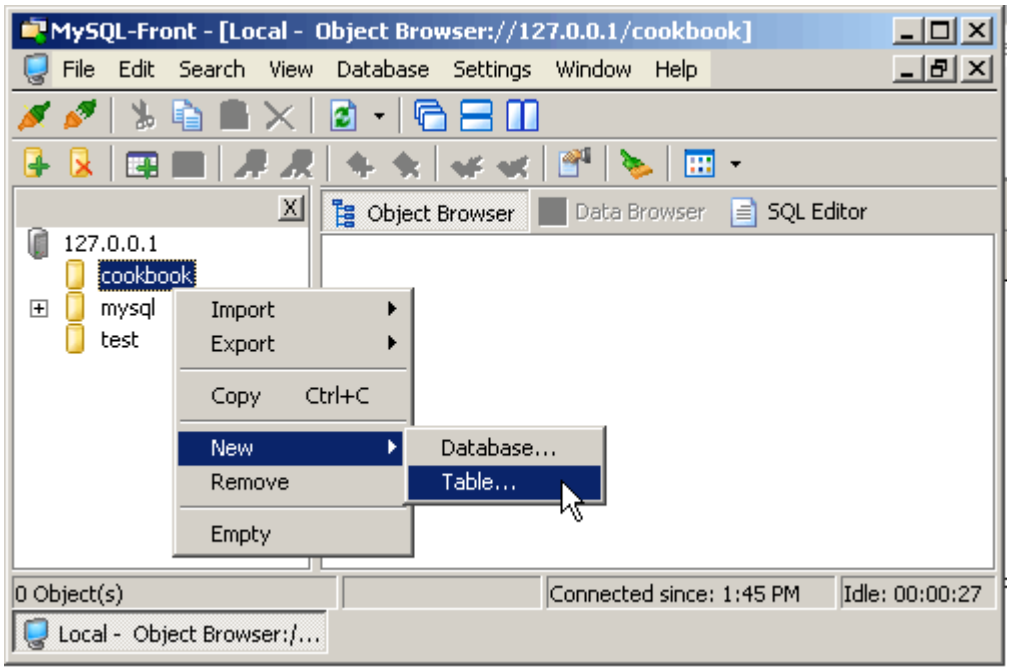

Рисунок 21. Создание новой таблицы

Назовите таблицу recipes (Рисунок 22).

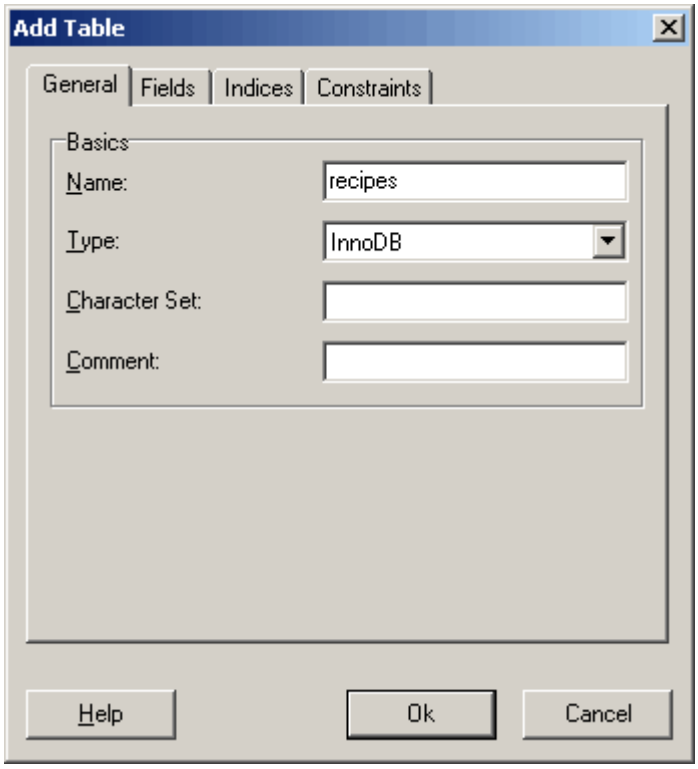

Рисунок 22. Окно добавления таблицы

Важная заметка: MySQL-Front автоматически создаст первичный ключ с именем `Id`, но Rails предпочитает называть его `id` (в нижнем регистре). Поэтому переименуйте его на `id`, кликнув правой кнопкой по полю `Id`, выберите Properties (Рисунок 23). ***(Прим. пер.**: в последней версии MySQL-Front первичный ключ назван уже в нижнем регистре)*

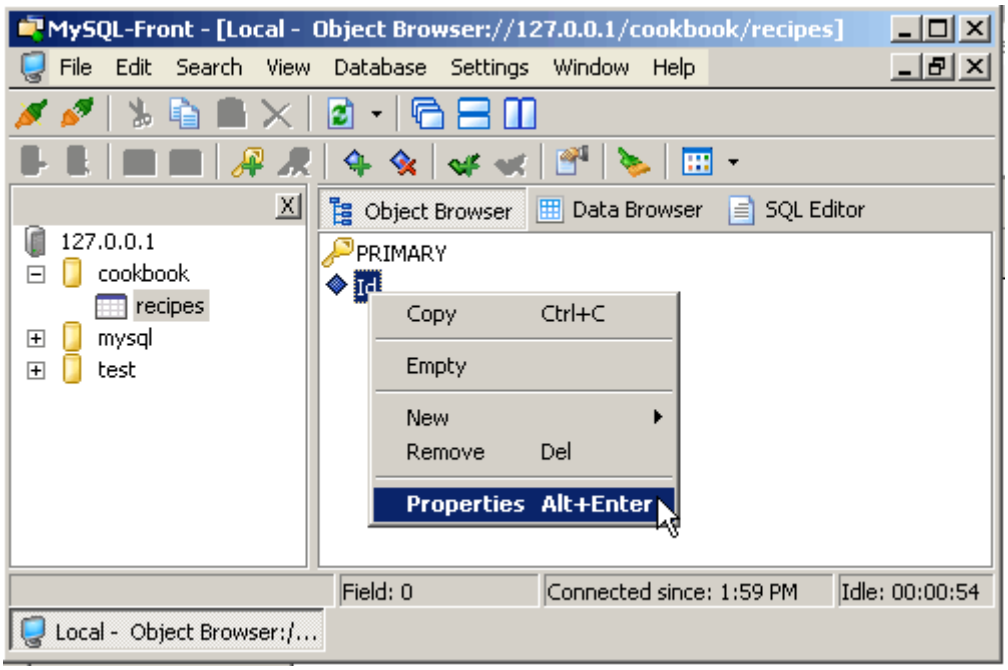

Рисунок 23. Переименование первичного ключа

Добавление полей

Теперь добавим поля, содержащие рецепты. Начнем с создания полей `title` и `instructions`.

При выбранной таблице recipes, щелкните правой кнопкой по пустой области правой панели и выберите `New>Field...` (Рисунок 24).

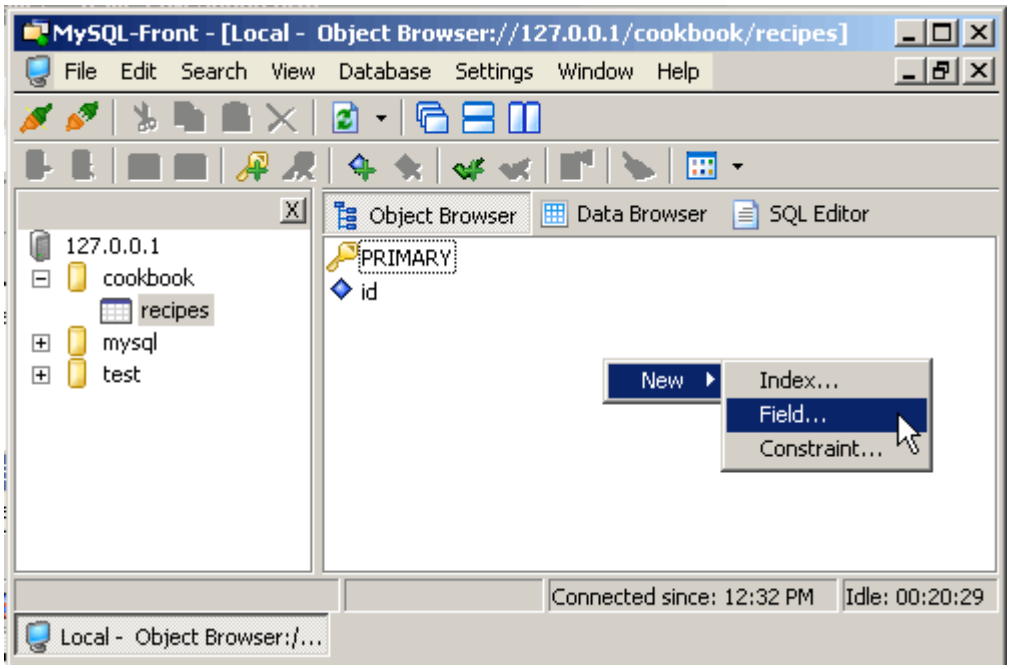

Рисунок 24. Добавление поля

Поле title – это ненулевой `varchar(255)`, то есть каждый рецепт должен иметь текстовую информацию в заголовке. См. Рисунок 25.

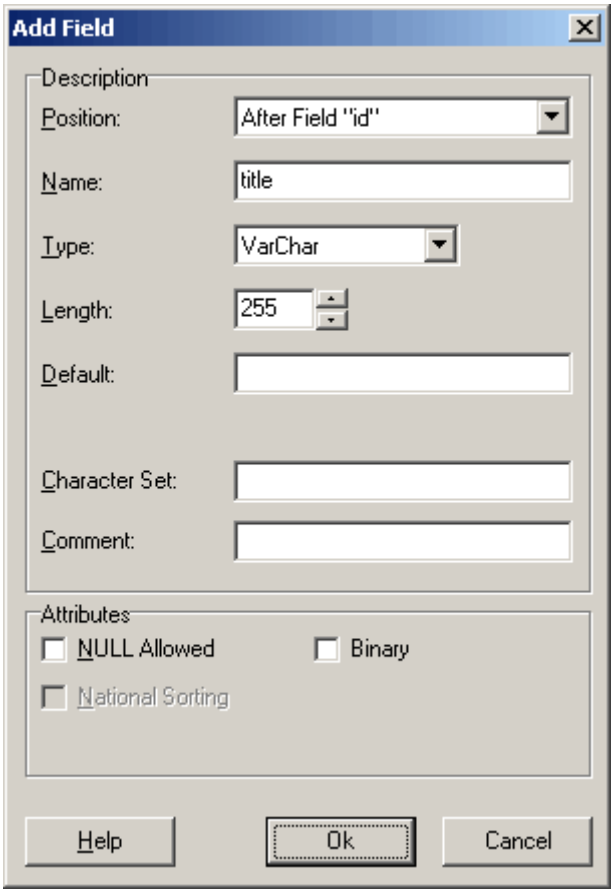

Рисунок 25. Добавление поля `title`

Повторите вышеприведенную процедуру для добавления поля `instructions` типа text, как показано на Рисунке 26.

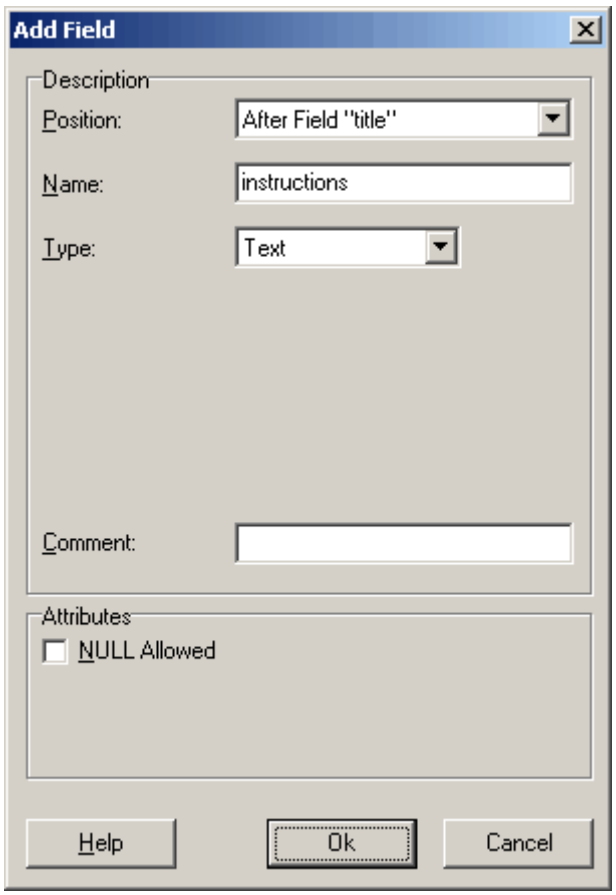

Рисунок 26. Добавление поля `instructions`

Наша таблица должна выглядеть так, как показано на Рисунке 27.

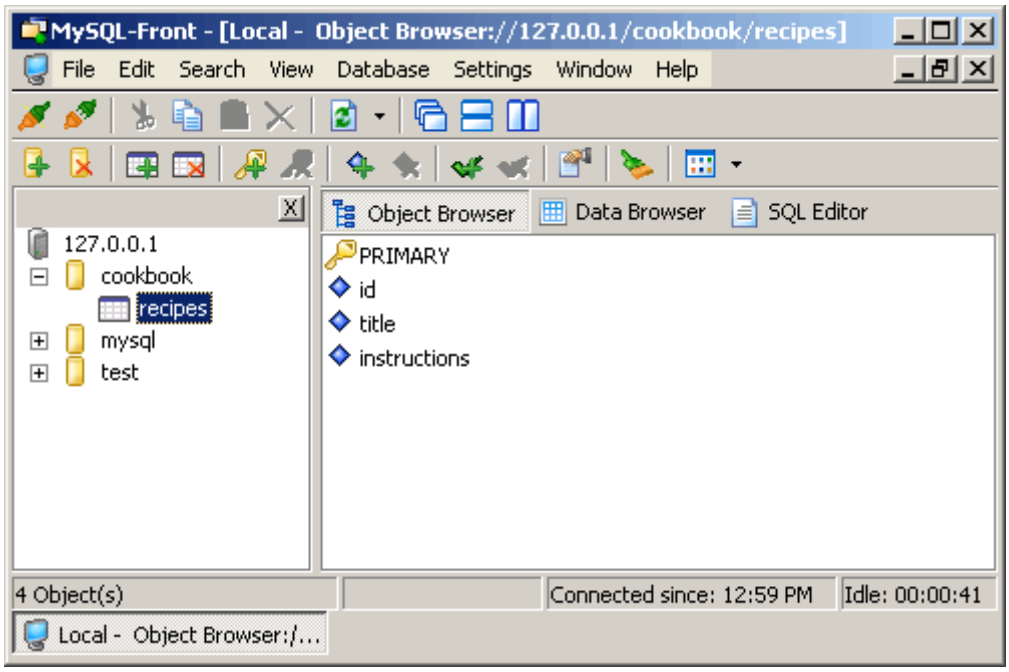

Рисунок 27. Таблица `recipe`

Возбуждение начинается:)

Все что мы, до этого сделали, было простым и безболезненным, но не выдающимся. Вот сейчас все и поменяется.

Создание модели

Во-первых, создадим модель `Recipe`, которая будет содержать данные из таблицы `recipes` базы данных. На рисунке 28 показано, где будет находиться файл.

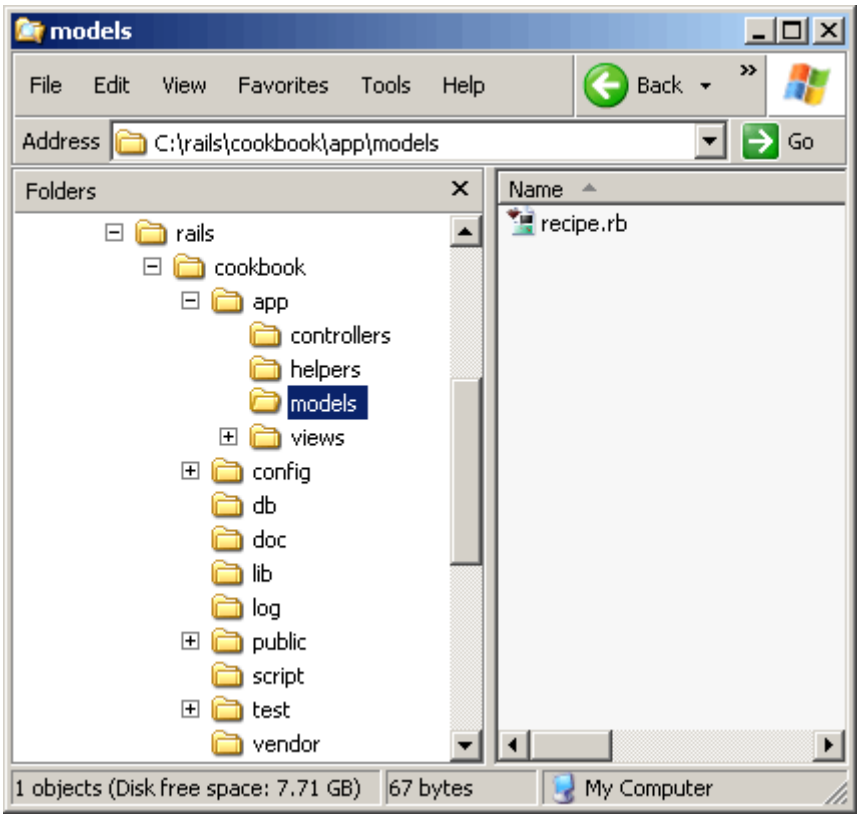

Рисунок 28. The `Recipe` model class

Откройте командную строку в каталоге (*c:\rails\cookbook*) и запустите команду:

```
ruby script\generate model Recipe
```

Это создаст файл *recipe.rb*, содержащий основу класса `Recipe`. Загляните внутрь файла (Рисунок 29).

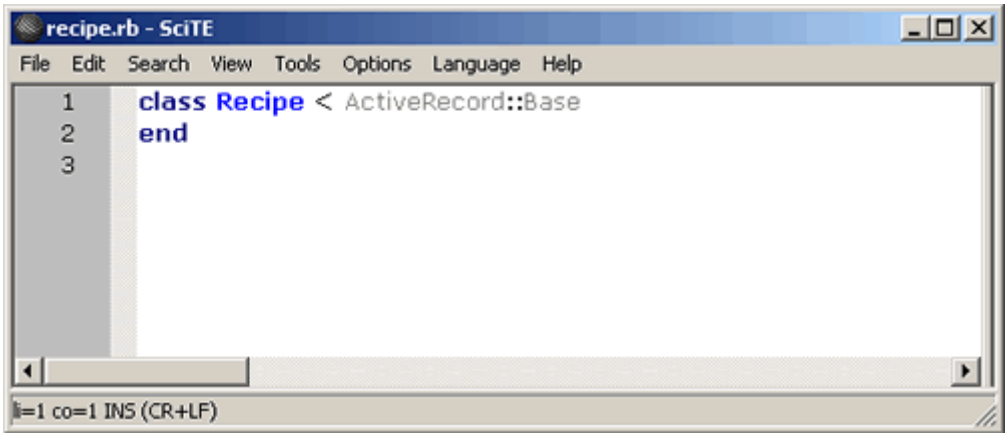

Рисунок 29. Содержимое файла recipe.rb

Пустое определение класса – объект, указывающий на таблицу `recipes` в базе данных. Более точно вы поймете, что я имею в виду буквально через минутку. А пока я хочу обратить внимание на некоторую магию, которая происходит в соглашении о наименовании в Rails: название класса модели в единственном числе (`Recipe`) указывает на базу данных, названную во множественном числе (`recipes`). Rails хорошо знает правила английского языка об единственном и множественном числах, поэтому `Company` указывает на `companies`, `Person` указывает на `people` и так далее.

Далее, Rails динамически наполняет класс `Recipe` методами доступа к рядам таблицы `recipes` и атрибутам каждой колонки.

Очень скоро вы увидите демонстрацию динамического соединения между классом `Recipe` и таблицей `recipes`.

Мы уже близки к тому, чтобы увидеть что-либо в работе. Нам необходимо создать контроллер recipe (Рисунок 30) с действиями для управления рецептами в нашей базе через стандартные операции CRUD: create (создание), read (чтение), update (обновление), и delete (удаление). Rails сделает это проще, чем вы думаете.

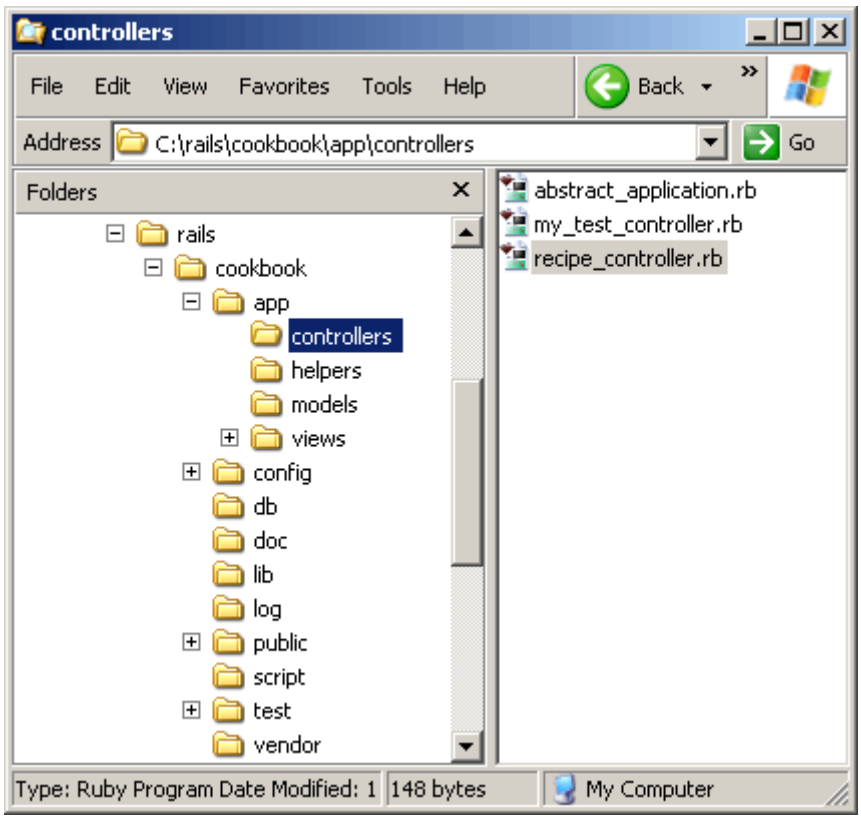

Рисунок 30. Контроллер Recipe

Откройте командную строку в каталоге (*c:\rails\cookbook*) и запустите команду:

```
ruby script\generate controller Recipe
```

При этом создастся файл *recipe_controller.rb* содержащий основу класса `RecipeController`. Откройте файл для редактирования и добавьте строку

```
scaffold :recipe
```

как показано на рисунке 31. (Что такое скаффолдинг?)

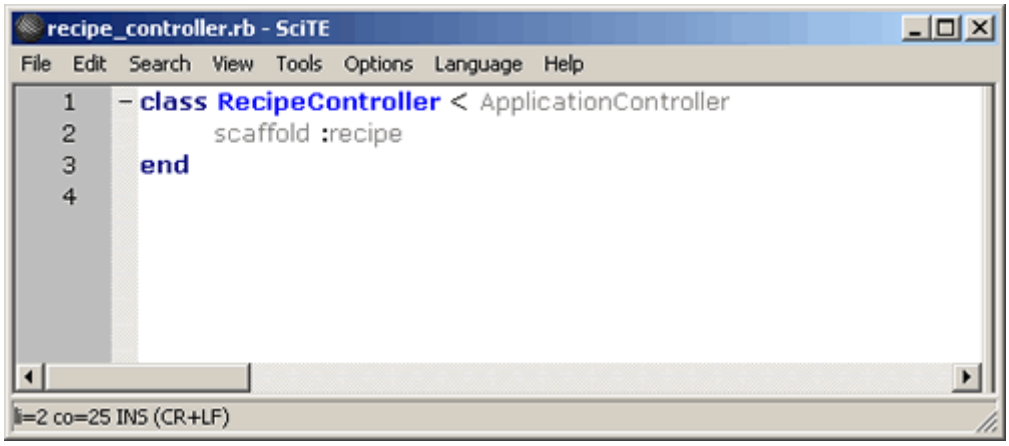

Рисунок 31. Код `RecipeController`

Эта единственная строка оживит нашу таблицу базы данных. Она немедленно определит действия для всех операций CRUD, позволив нам создавать, читать, обновлять и удалять рецепты в базе данных!

Откройте браузер и перейдите на http://127.0.0.1:3000/recipe/new. См. Рисунок 32.

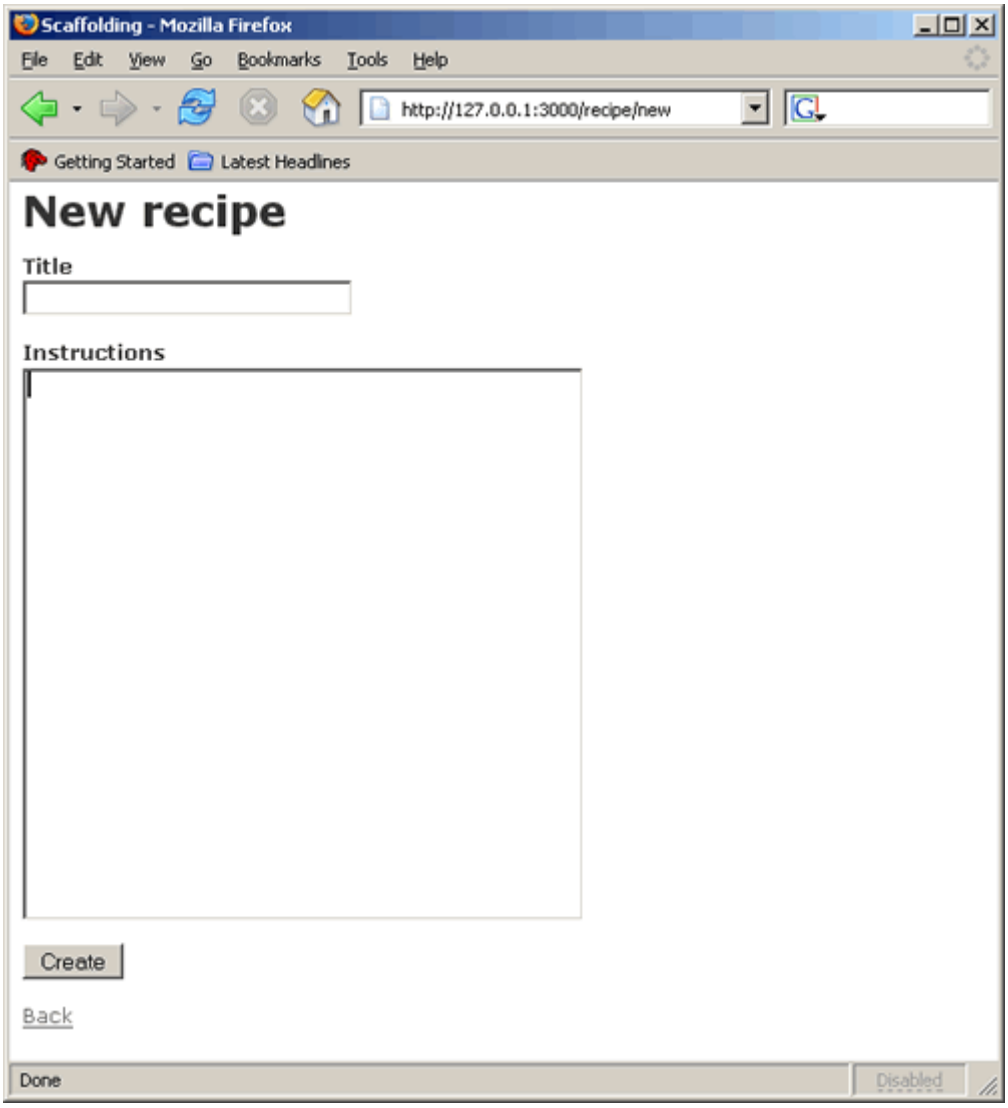

Рисунок 30. Creating a new recipe page

Вот это круто! Мы почти ничего не сделали, а уже может наполнять нашу базу данных. Но пока не делайте это, а давайте сначала добавим еще полей в таблицу `recipe`.

Используйте MySQL-Front, чтобы добавить поля `description` и `date` между `title` и `instructions` (Рисунки 33 и 34).

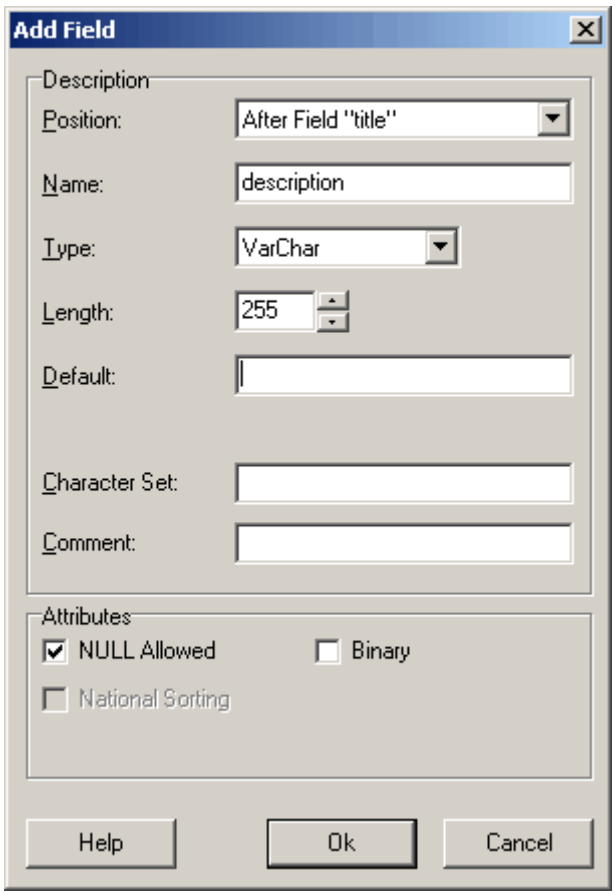

Рисунок 33. Добавление поля `description`

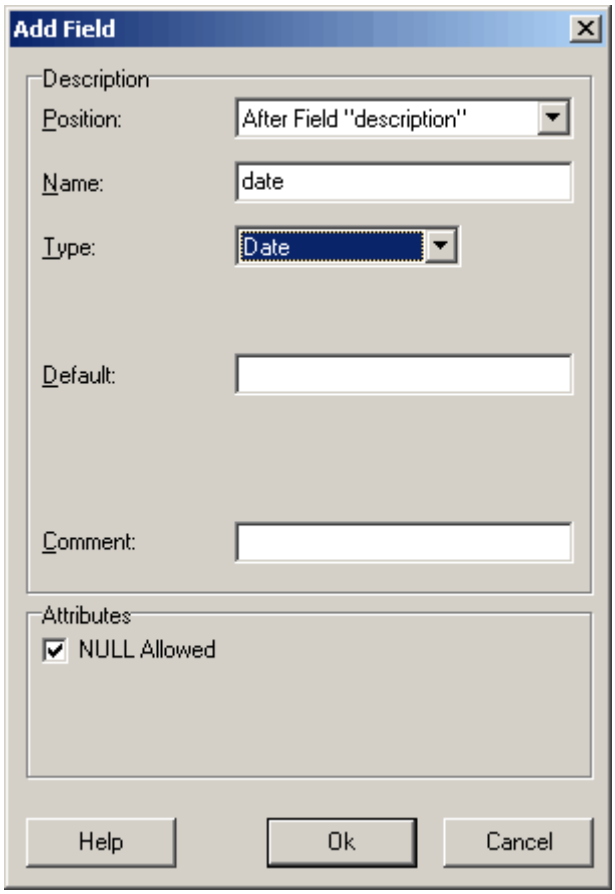

Рисунок 34. Добавление поля `date`

Обновим страницу. См. Рисунок 35.

Рисунок 35. A new recipe page with the new fields

Вот это красиво!

Хорошо, успокоимся и добавим тестовый рецепт. Заполним поля как показано на Рисунке 36 и нажмем кнопку Create.

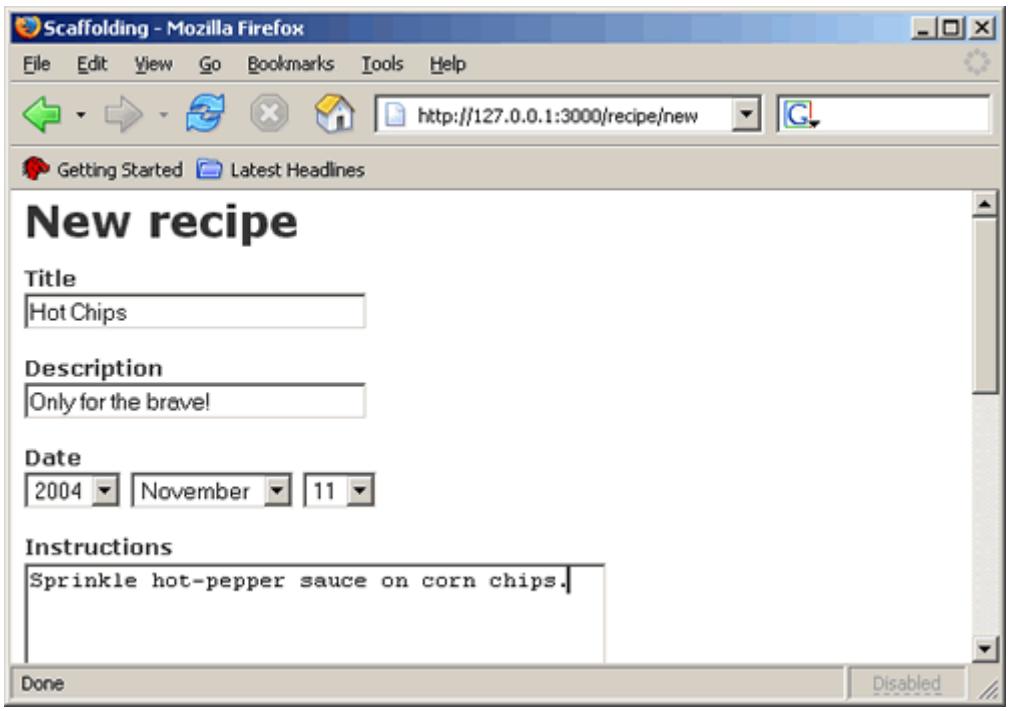

Рисунок 36. Новый рецепт

Результат показан на Рисунке 37.

Рисунок 37. Список всех рецептов

Добавьте еще один рецепт, нажав на ссылку "New recipe" и введя данные, как показано на Рисунке 38.

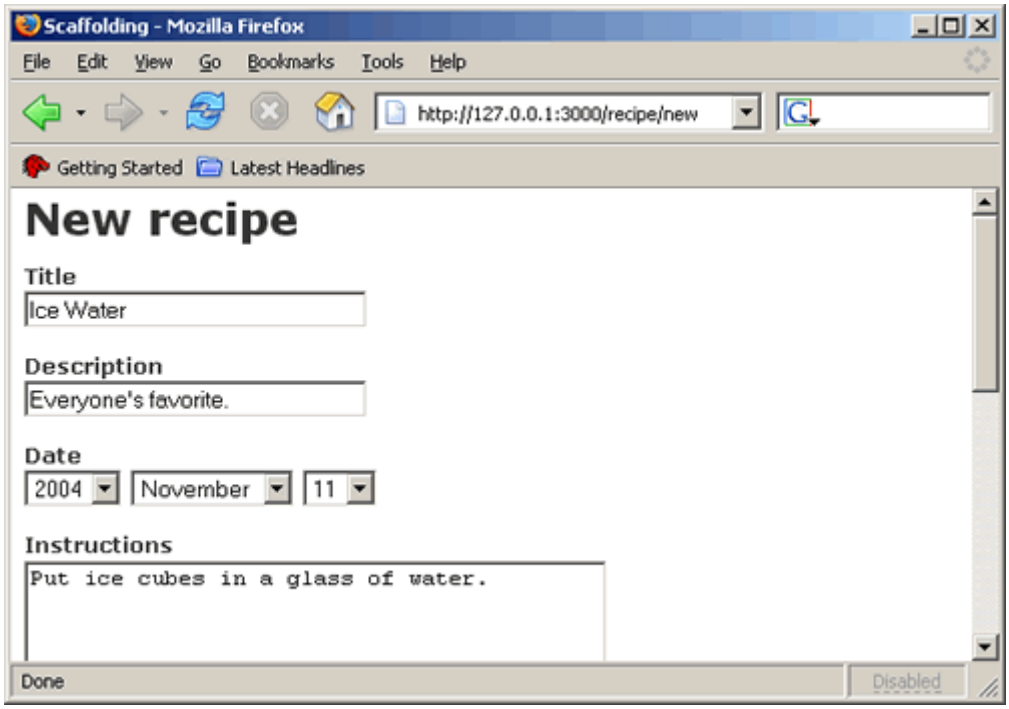

Рисунок 38. Другой рецепт

После нажатия на Create вы увидите что-то наподобие Рисунка 39.

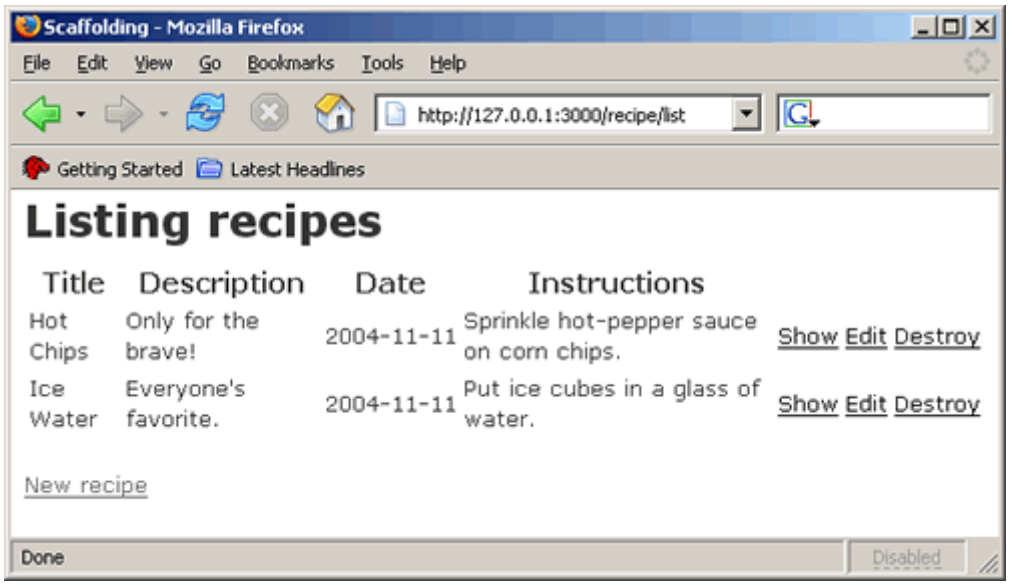

Рисунок 39. Список рецептов

Теперь у нас есть куча функционала для базы данных, содержащегося в одной строке кода. Внешний вид конечно пока не очень, но мы исправим это скоро.

Что же произошло?

Одна строка кода `scaffold :recipe` оживила все. Она позволила вам начать работу с нашей моделью данных. Без помощи с нашей стороны были созданы действия `list`, `show`, `edit`, и `delete`. А также были созданы темплейты по умолчанию для каждого из этих действий.

Конечно все действия и внешний вид пока простые – никакой сортировки и т.п. Теперь оставим скаффолдинг в покое и создадим собственные версии видов и действий. Каждый раз при создании собственного вида или действия, вы переопределяете их версию в скаффолдинге. Когда вы все сделали, версия скаффолдинга просто удаляется из контроллера.

Заметили ли вы имена URL? Rails пытается создавать красивые URL. Они просты и указывают куда надо, а не длинные и зашифрованные.

Создание действий и видов

Страница, отображающая рецепты, нуждается в улучшении. Чтобы сделать это, необходимо переопределить версию скаффолдинга для действия `list`.

Откройте файл *recipe_controller.rb* и добавьте метод `list`. См. Рисунок 40.

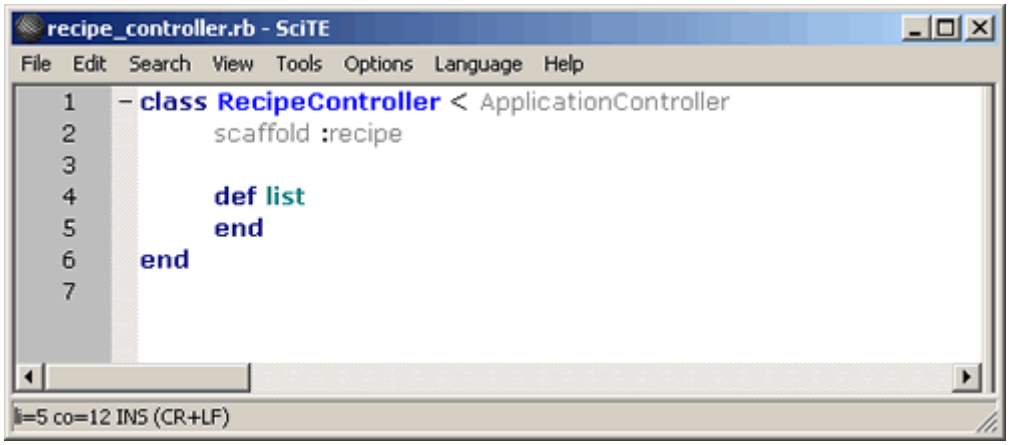

Рисунок 40. Новый метод `list`

Перейдите на `http://127.0.0.1:3000/recipe/list`. См. Рисунок 41.

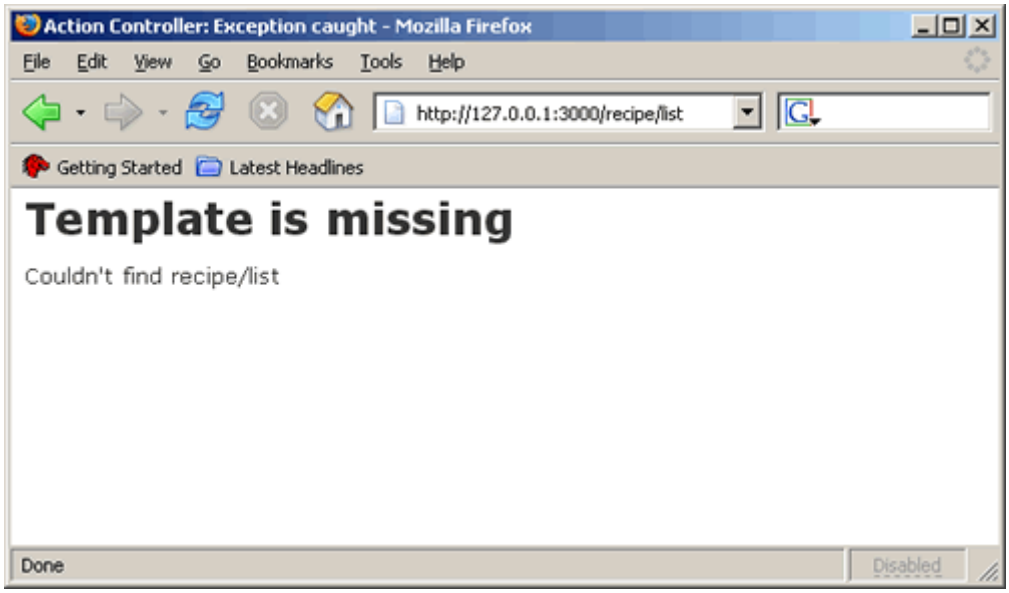

Рисунок 41. Результаты нового метода `list`

Так как вы создали собственное определение действия `list`, Rails больше не использует версию скаффолдинга. Rails вызывает наш метод `list` и пытается найти темплейт вида,

чтобы его отобразить. Так мы его еще не создали, то получаем ошибку "template missing". Давайте создадим этот темплейт для действия `list`, отображающий заголовок рецепта и дату.

При создании контроллера recipe, скрипт `generate` также создал каталог видов, куда мы можем помещать HTML темплейты для отображения. В каталоге *c:\rails\cookbook\app\views\recipe* создадим файл *list.rhtml*. Если вы работали с JSP или ASP страницами, это будет для вас знакомо. Это простой html файл с кодом Ruby, заключенным в теги `<% %>` и `<%= %>`.

Файл *list.rhtml* содержит следующий код:

```
<html>
<head>
<title>All Recipes</title>
</head>
<body>

<h1>Online Cookbook - All Recipes</h1>
<table border="1">
 <tr>
  <td width="80%"><p align="center"><i><b>Recipe</b></i></td>
  <td width="20%"><p align="center"><i><b>Date</b></i></td>
 </tr>

 <% @recipes.each do |recipe| %>
  <tr>
   <td><%= link_to recipe.title, :action => "show", :id => recipe.id %></td>
   <td><%= recipe.date %></td>
  </tr>
 <% end %>
</table>
<p><%= link_to "Create new recipe", :action => "new" %></p>

</body>
</html>
```

Откройте файл *recipe_controller.rb* и добавьте в метод list одну строку кода. См. Рисунок 42.

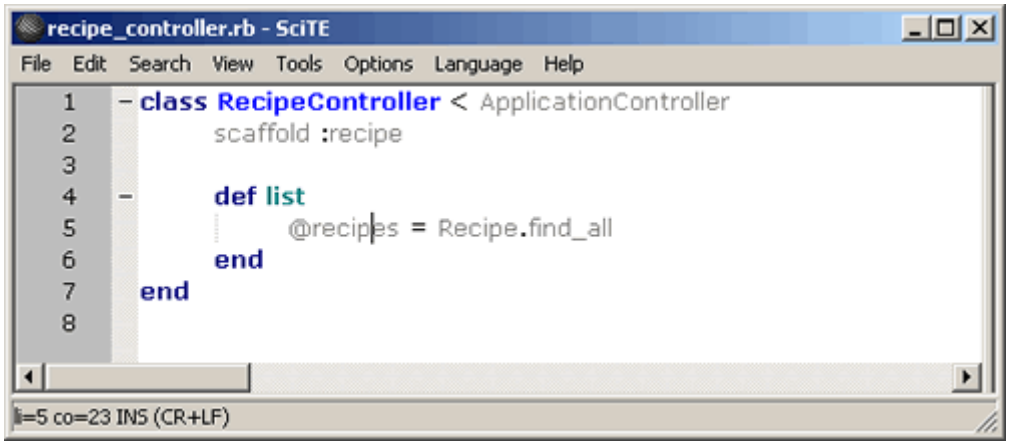

Рисунок 42. Код для отображения всех рецептов

Обновим страницу браузера. См. Рисунок 43.

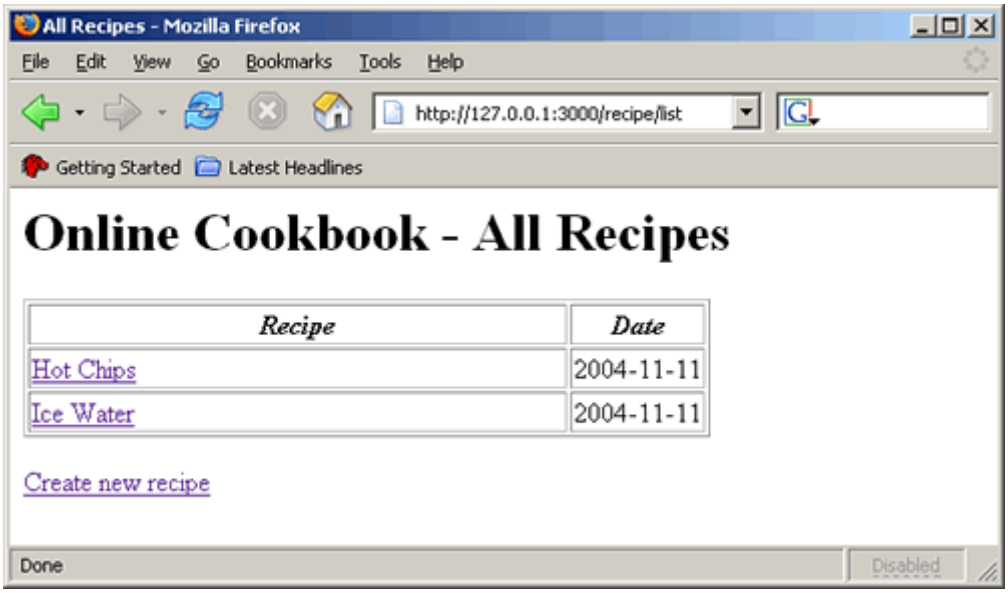

Рисунок 43. Более красивое отображение рецептов

Определенно лучше! Как же это работает?

```
def list
  @recipes = Recipe.find_all
end
```

Когда вы переходите на http://127.0.0.1:3000/recipe/list, Rails вызывает новый метод `list`. Одна строка кода запрашивает у класса `Recipe` коллекцию всех рецептов из базы данных, назначив эту коллекцию переменной `@recipes`.

Затем, Rails ищет темплейт этого действия и отображает его в браузере. Большая часть темплейта – это обычный HTML. Настоящие чудеса происходят здесь:

```
<% @recipes.each do |recipe| %>
  <tr>
   <td><%= link_to recipe.title, :action => "show", :id => recipe.id %></td>
   <td><%= recipe.date %></td>
  </tr>
 <% end %>
```

Этот код Ruby проходит через всю коллекцию рецептов, извлеченную из контроллера. Первая ячейка содержит ссылку на страницу отображения рецепта. Обратите внимание на атрибуты, используемые в объекте recipe (`title`, `id`, и `date`). Они являются полями таблицы `recipes`.

Добавление категорий в кулинарную книгу

Теперь нам надо добавить возможность назначать категорию для каждого рецепту (например "dessert") и отображать только рецепты определенной категории. Чтобы это сделать, необходимо добавить таблицу категорий в базу данных и поле в таблице рецептов, содержащее информацию о категории, к которой принадлежит рецепт.

В MySQL-Front создайте таблицу `categories`. Не забудьте поменять `Id` на `id`. Создайте поле `name` - `varchar(50)`. См. Рисунок 44.

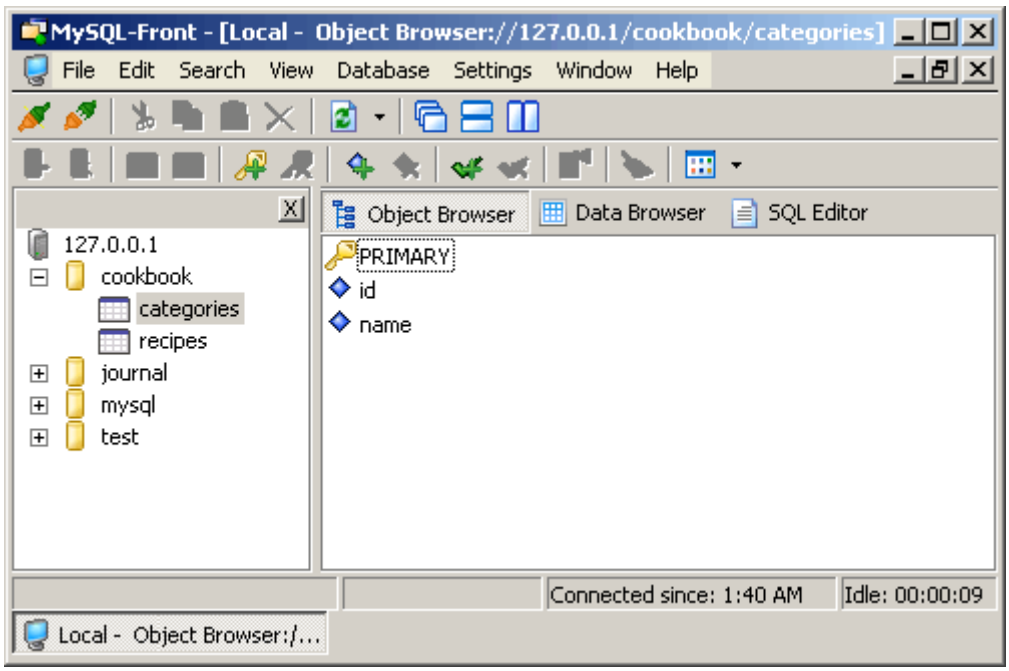

Рисунок 44. Таблица `categories`

Нам также нужны контроллер и модель для категорий. Откройте командную строку в каталоге cookbook и выполните две команды (Рисунок 45):

```
ruby script\generate controller Category
ruby script\generate model Category
```

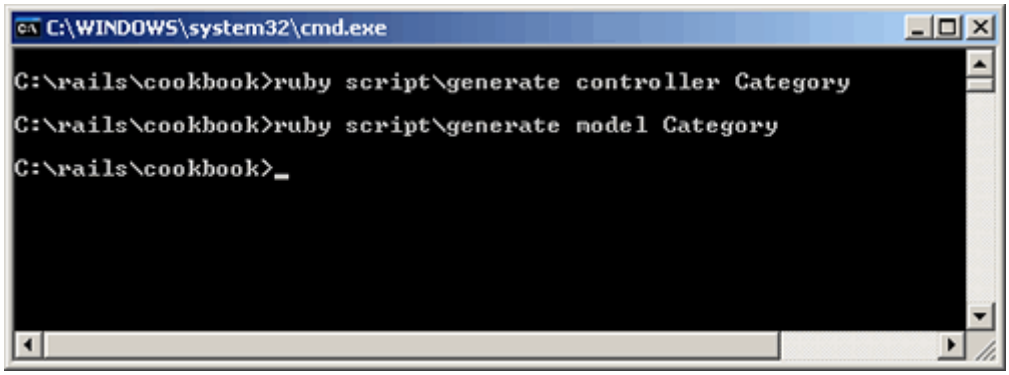

Рисунок 45. Создание модели и контроллера категорий

Теперь добавим скаффолдинг в контроллер категорий. Откройте *c:\rails\cookbook\app\controllers\category_controller.rb* и добавьте строку со скаффолдингом. См. Рисунок 46.

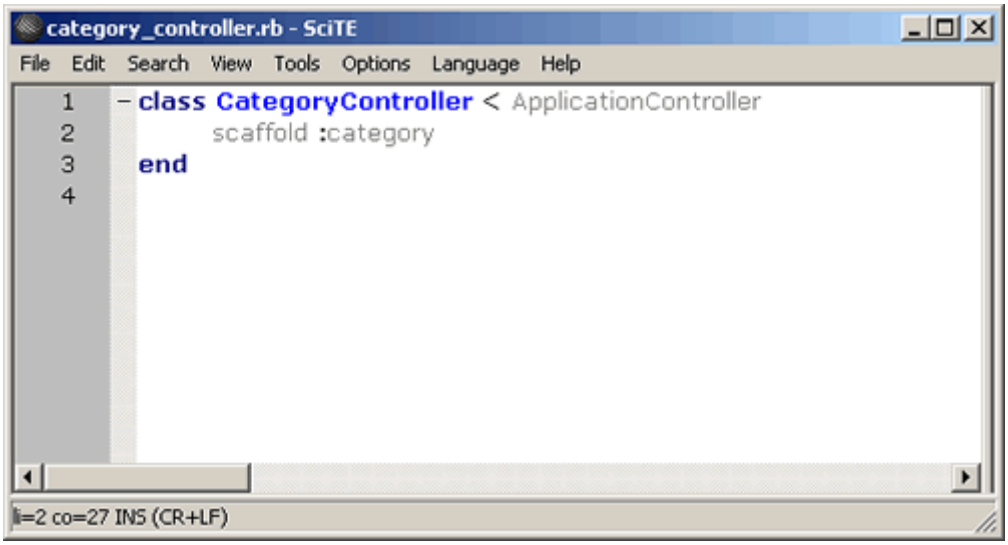

Рисунок 46. Добавление скаффолдинга

Перейдем на `http://127.0.0.1:3000/category/new` и создадим две категории `Snacks` и `Beverages`. См. Рисунок 47.

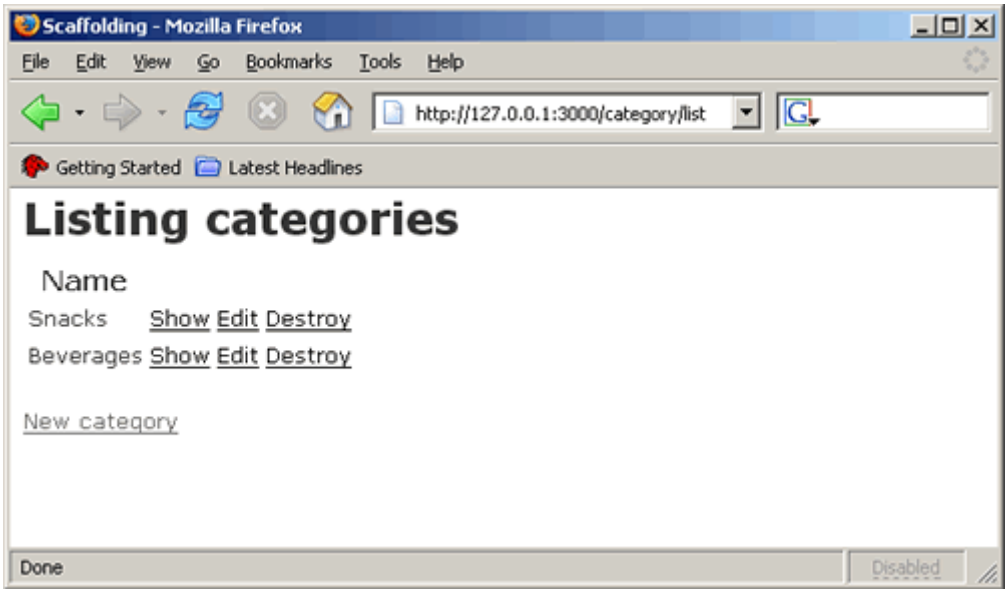

Рисунок 47. Список всех категорий

Назначение категории рецепту

Теперь кулинарная книга имеет рецепты и категории и их необходимо связать вместе. Мы хотим назначать категорию рецепту. Для этого необходимо добавить поле в таблицу recipes, содержащее `id` категории, а потом создать действие `edit`, содержащее выпадающий список категорий.

Во-первых, добавим поле `category_id` в таблицу `recipe` - `int(6)`, соответствующее ключу таблицы `category`. См. Рисунок 48.

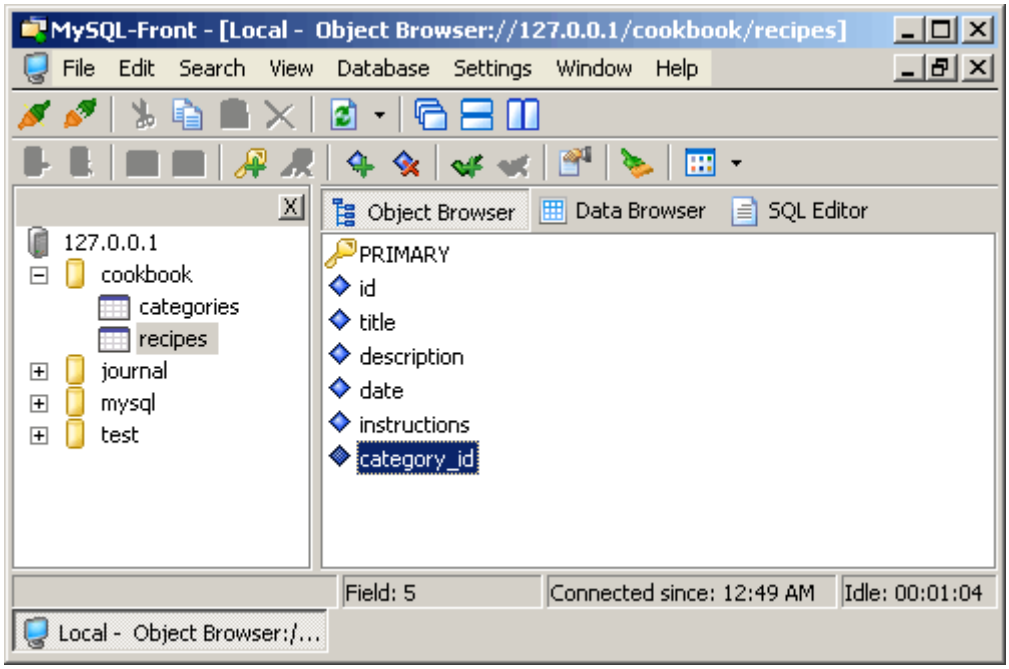

Рисунок 48. Таблица `recipe` с новым полем `category_id`

Поле будет содержать `id` категории. Теперь расскажем модели `Recipe` об этом.

Откройте *c:\rails\cookbook\app\models\recipe.rb* и *c:\rails\cookbook\app\models\category.rb* и добавьте следующие строки кода соответственно. См. рисунки 49 и 50:

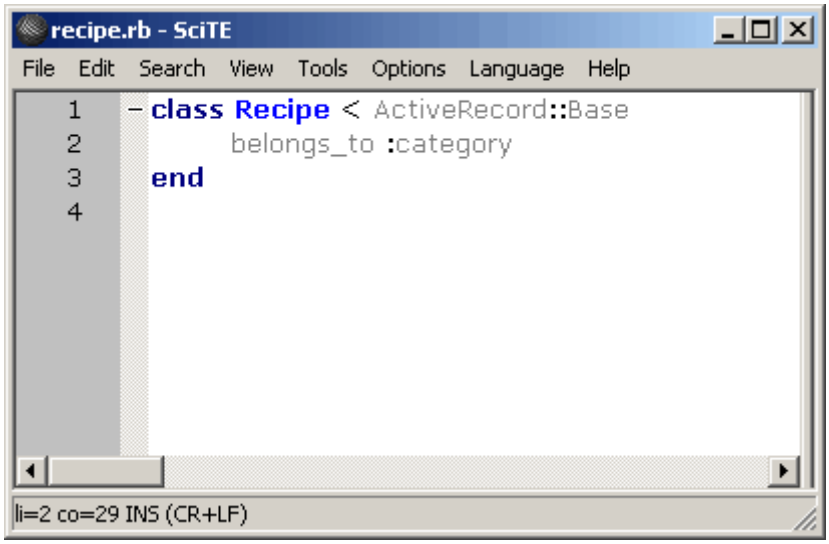

Рисунок 49. Установка отношений в модели `Recipe`

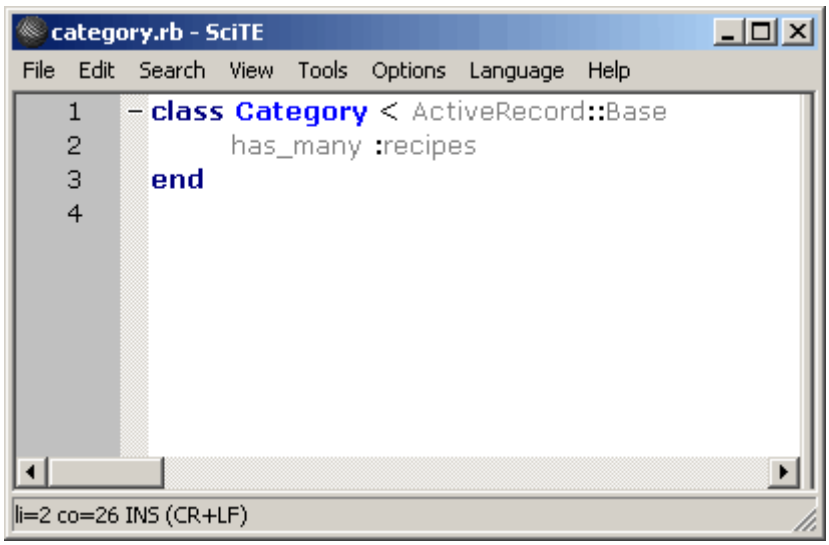

Рисунок 50. Установка отношений в модели `Category`

Это говорит Rails, что рецепт принадлежит категории, а категория может иметь много рецептов. Эти записи генерируют методы для этих взаимоотношений в коде Ruby.

Например, если у вас объект рецепта в `@recipe`, можно найти название его категории с помощью кода `@recipe.category.name`. Также, если объект категории в `@category`, можно получить коллекцию всех рецептов этой категории с помощью кода `@category.recipes`.

Теперь настало время создать действие `edit` и его темплейт. Откройте *c:\rails\cookbook\app\controllers\recipe_controller.rb* и добавьте метод `edit`. См. Рисунок 51.

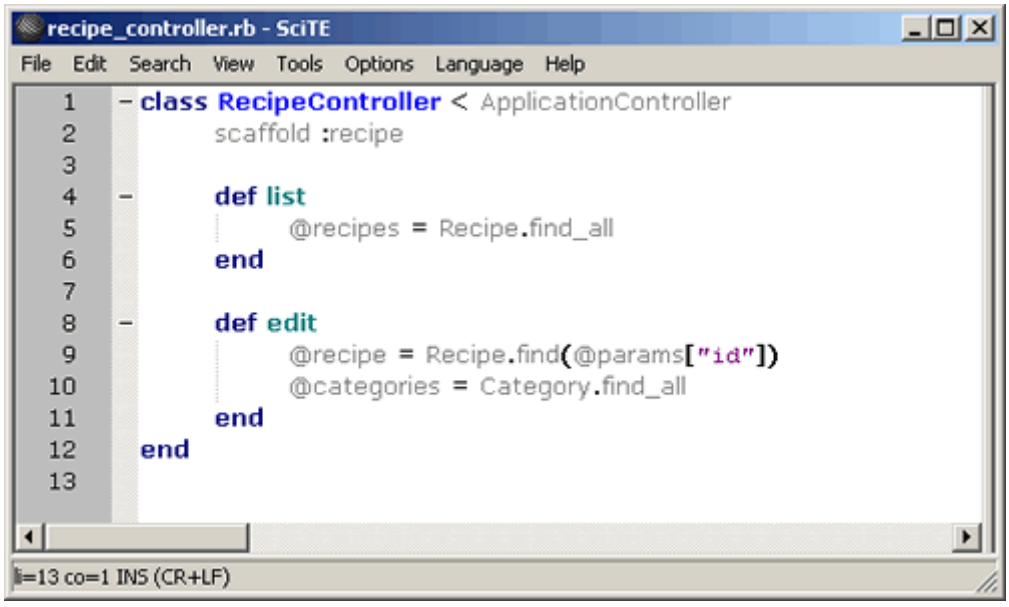

Рисунок 51. Метод `edit`

При этом создаются две переменные, которые темплейт будет использовать при отображении страницы редактирования рецептов. `@recipe` – это рецепт, который мы хотим редактировать (параметр `id` приходит из веб-запроса). `@categories` – это коллекция всех категорий в базе данных. Темплейт будет использовать ее для создания выпадающего списка категорий.

В каталоге *c:\rails\cookbook\app\views\recipe* создайте файл *edit.rhtml*, содержащий HTML темплейт, показанный ниже. В основном это стандартный HTML с основным кодом Ruby в тегах `<select>` и `<option>`:

```
<html>
 <head>
  <title>Edit Recipe</title>
 </head>
 <body>
 <h1>Edit Recipe</h1>

 <form action="../update/<%= @recipe.id %>" method="POST"">
  <input id="recipe_id" name="recipe[id]" size="30"
         type="hidden" value="<%= @recipe.id %>" />
  <p><b>Title</b><br>
  <input id="recipe_title" name="recipe[title]" size="30"
         type="text" value="<%= @recipe.title %>" />
  </p>
  <p><b>Description</b><br>
  <input id="recipe_description" name="recipe[description]"
         size="30" type="text"
         value="<%= @recipe.description %>" />
  </p>
  <p><b>Category:</b><br>

  <select name="recipe[category_id]">
   <% @categories.each do |category| %>
       <option value="<%= category.id %>"
         <%= ' selected' if category.id == @recipe.category_id %>>
         <%= category.name %>
       </option>
   <% end %>
  </select></p>

  <p><b>Instructions</b><br>
  <textarea cols="40" id="recipe_instructions"
            name="recipe[instructions]"
            rows="20" wrap="virtual">
    <%= @recipe.instructions %>
  </textarea> </p>
  <input type="submit" value="Update" />
 </form>

 <a href="/recipe/show/<%= @recipe.id %>">
   Show
 </a> |
  <a href="/recipe/list">
  Back
 </a>

 </body>
</html>
```

Здесь используются переменные `@recipe` и `@categories`. Обратите внимание на место, где циклом проходим через все категории, чтобы создать выпадающий список. Посмотрите на тег `<option> и обратите внимание, как` он использует назначенную категорию. Изучите пример и затем попробуйте его запустить.

Перейдите на http://127.0.0.1:3000/recipe/list и отредактируйте рецепт "Ice Water." Измените категорию на "Beverages". См. Рисунок 52.

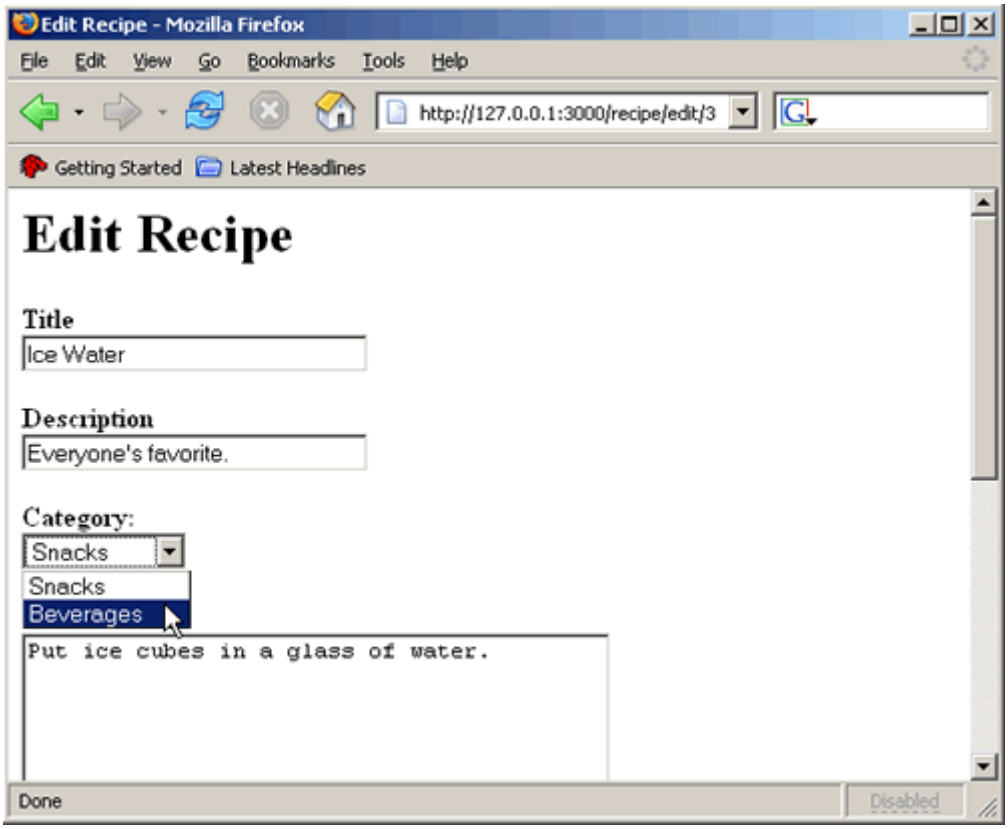

Рисунок 52. Изменение категории рецепта

Перед последним шагом убедитесь, что все рецепты имеют категории. Добавьте категорию и обновите каждый рецепт. Если вы этого не сделаете, то дальше возникнут ошибки.

Отображение категорий в списке рецептов

Это последний шаг. Изменим темплейт отображения списка, чтобы выводились категории.

Отредактируйте файл *c:\rails\cookbook\app\views\recipe\list.rhtml* следующим образом:

```
<html>
<head>
<title>All Recipes</title>
</head>
<body>

<h1>Online Cookbook - All Recipes</h1>
<table border="1">
 <tr>
  <td width="40%"><p align="center"><i><b>Recipe</b></i></td>
  <td width="20%"><p align="center"><i><b>Category</b></i></td>
  <td width="20%"><p align="center"><i><b>Date</b></i></td>
 </tr>

 <% @recipes.each do |recipe| %>
  <tr>
   <td><%= link_to recipe.title, :action => "show", :id => recipe.id %></td>
   <td><%= recipe.category.name %></td>
   <td><%= recipe.date %></td>
  </tr>
 <% end %>
```

```
</table>
<p><%= link_to "Create new recipe", :action => "new" %></p>

</body>
</html>
```

Теперь перейдите на http://127.0.0.1:3000/recipe/list. См. Рисунок 53.

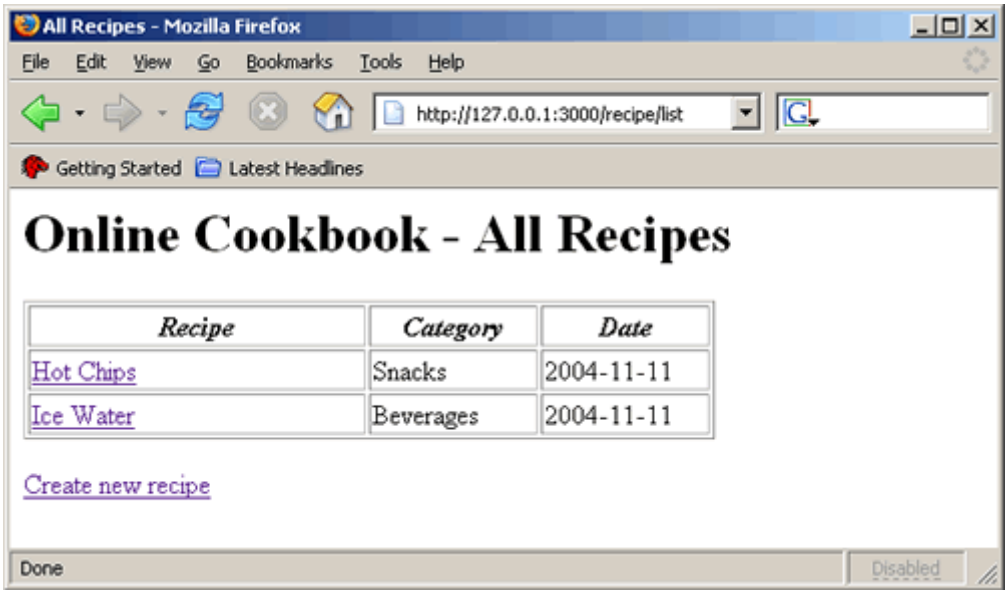

Рисунок 53. Рецепты с категориями

Домашнее задание

Поздравляю, вы создали Rails приложение! Оно, конечно, требует доработки, но зато работает.

Вот домашнее задание для вас:

- Теперь нет способа, чтобы удалить рецепт. Добавьте его.
- На главной странице нет ссылок на страницу управления категориями. Добавьте ее.
- Было бы хорошо отображать рецепты конкретной категории. Например, мы хотим видеть рецепты всех чипсов или напитков. На странице списка рецептов добавьте ссылки к категориям на страницы отображения рецептом только этих категорий.

Во второй части этой статьи мы выполним эти задачи.

Rolling with Ruby on Rails, часть 2 и Ajax on Rails на английском..

Ресурсы

Сайты

- Оффициальная домашняя страница Ruby
- Оффициальная домашняя страница Ruby on Rails
- Бесплатная книга *Programming Ruby*
- Местоположение open source Ruby проектов: RubyForge
- Интернет-журнал Дмитрия Сабанина

Мейл-листы

- Rails mailing list
- Ruby-talk mailing list

Curt Hibbs is a senior software developer in St. Louis with more than 30 years' experience in platforms, languages, and technologies too numerous to list.