Extracted from:

Rails Recipes

Rails without a Database

Problem

"Opinionated software" as it is, Rails assumes you want to develop with a database. This is *usually* the case, which is the reason the assumption. But what if you're developing an application with a file-based back end? Or perhaps you're simply front-ending an external service-based API. Rails is a little less friendly to you in this case—particularly when testing your application.

Solution

By default, Rails assumes that you want to connect to and initialize a database whenever you run your tests. This means that if you don't have a database, testing is difficult to do. Of course, you could just create a database for nothing, but that would mean you'd have extra infrastructure to support for no reason. A little hacking on a generated Rails application will get it into testable shape without a database.

To keep things simple and repeatable, we'll start with a fresh application. You'll be able to easily apply what we do here to your own application. Let's generate an application now. You can call it whatever you like. Mine is named DatabaselessApplication.

Next we'll create a simple class in lib for which to write some tests. Let's be really simple and create a class called Adder that adds numbers together:

Download DatabaselessApplication/lib/adder.rb

```
class Adder
  def initialize(first, second)
    @first = first
    @second = second
  end

  def sum
    @first + @second
  end
end
```

Now we'll create a simple test case for it in test/unit/adder_test.rb:

Download DatabaselessApplication/test/unit/adder_test.rb

```
require File.join(File.dirname(__FILE__), "..", "test_helper")
require 'adder'
class AdderTest < Test::Unit::TestCase
  def test_simple_addition
    assert_equal(4, Adder.new(3,1).sum)
  end
end
```

Let's try to run the test:

```
chad> rake test:units
   (in /Users/chad/src/FR_RR/Book/code/DatabaselessApplication)
rake aborted!
#42000Unknown database 'databaselessapplication_development'
```

It seems that the Rails test:units() Rake task does some database initialization. In fact, rake -P confirms this:

```
chad> rake -P
 ...
rake test:units
    db:test:prepare
 ...
```

Sure enough, test:units() depends on the db:test:prepare() task. What if we tried to run the tests directly, not using our Rake task?

```
chad> ruby test/unit/adder_test.rb
Loaded suite test/unit/adder_test
Started
EE
Finished in 0.052262 seconds.

  1) Error:
test_simple_addition(AdderTest):
Mysql::Error: #42000Unknown database 'databaselessapplication_test'
(abbreviated)
```

Some digging shows that somewhere in the chain of having required test_helper.rb, we inherited the database-centric setup() and teardown() methods. We could just use require "test/unit" manually right here in the test, but then we'd have to replicate this in every test we create. We would also find that this wouldn't initialize the Rails environment as necessary. So instead, we'll modify test_helper.rb itself.

Specifically, test_helper.rb's inclusion of test_help.rb is the source of the problem. So instead of the require() call to test_help.rb, we'll just cherry-pick what we want from it. And since we're removing the fixture-related

definitions, we'll remove all of the generated fixture-related code as well. Here's our new test_helper.rb:

Download DatabaselessApplication/test/test_helper.rb

```
ENV["RAILS_ENV"] = "test"
require File.expand_path(File.dirname(__FILE__) + "/../config/environment")
require 'application'
require 'test/unit'
require 'action_controller/test_process'
require 'action_web_service/test_invoke'
require 'breakpoint'
```

If you don't plan to use ActionWebService, it's safe to remove the line that requires action_web_service/test_invoke.

Running our test as we did before now passes!

```
chad> ruby test/unit/adder_test.rb
Loaded suite test/unit/adder_test
Started
.
Finished in 0.002703 seconds.

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

Now that we have a unit test working, let's try a functional test. We'll first need to generate a controller (and the related tests):

```
chad> ruby script/generate controller MyController
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/my_controller
      create  test/functional/
      create  app/controllers/my_controller_controller.rb
      create  test/functional/my_controller_controller_test.rb
      create  app/helpers/my_controller_helper.rb
```

Let's just try to run this test as is. Maybe it'll work:

```
chad> ruby test/functional/my_controller_controller_test.rb
Loaded suite test/functional/my_controller_controller_test
Started
.
Finished in 0.002624 seconds.

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

Well, that was easy, wasn't it? All that's left is to get these tests working with Rake. Having to manually invoke our test files one at a time is a real step backward from the default Rails way of testing. The Rails built-in testing tasks work really well, so we'd rather not lose any functionality as we implement our own tasks. We also don't want to have to copy and paste their code into our own Rake tasks. If we did that, we wouldn't receive the benefits of bug fixes and upgrades to the built-in tasks. If only the built-in tasks didn't have that db:test:prepare() prerequisite!

Fortunately, with the way Rails loads user-defined Rake tasks, we aren't limited to simply defining our own *new* tasks. We can also use drop-in Rakefiles to modify the behavior of the built-in Rake tasks before they are executed. The three tasks we're specifically interested in are test:units(), test:functional(), and recent() (a really handy task that runs only those tests that have recently changed). If we create the following file in lib/tasks/clear_database_prerequisites.rake, it will do the trick for us:

Download DatabaselessApplication/lib/tasks/clear_database_prerequisites.rake

```
["test:units", "test:functionals", "recent"].each do |name|
  Rake::Task[name].prerequisites.clear
end
```

If you're running Rails 1.1 or higher, you'll need to add the task test:integration to the list of prerequisites to clear.

Rake 0.7.0 introduced a slight incompatibility in its API. If you're using a Rake version *prior to* 0.7.0, you'll need this instead: And, if you're running Rails 1.1 or higher, you'll need to add test:integration to the list of prerequisites.

Download DatabaselessApplication/lib/tasks/clear_database_prerequisites.rake

```
["test:units", "test:functionals", "recent"].each do |name|
  Rake::Task.lookup(name).prerequisites.clear
end
```

This single line looks up each Task using Rake's API and clears the Task's

dependencies. With this file installed, we can now successfully run any of the three built-in testing tasks without a database!

Discussion

Though it's not necessary to get our application running without a database, we can save memory and improve performance by controlling which Rails frameworks get loaded when our applications initialize. In a freshly generated Rails 1.0 application, the file config/environment.rb contains a section that looks like the following:

```
# Skip frameworks you're not going to use
# config.frameworks -= [ :action_web_service, :action_mailer ]
```

If we uncomment this line, we can specify any of the frameworks that we *do not* plan on using. For this recipe, it would make sense to add :active_record to this list.

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help programmers stay on top of their game.

Visit Us Online

Rails Recipes Home Page
`pragmaticprogrammer.com/titles/fr_rr`
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
`pragmaticprogrammer.com/updates`
Be notified when updates and new books become available.

Join the Community
`pragmaticprogrammer.com/community`
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
`pragmaticprogrammer.com/news`
Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragmaticprogrammer.com/titles/fr_rr.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragmaticprogrammer.com/catalog
Customer Service:	support@pragmaticprogrammer.com
Non-English Versions:	translations@pragmaticprogrammer.com
Pragmatic Teaching:	academic@pragmaticprogrammer.com
Author Proposals:	proposals@pragmaticprogrammer.com