Rails Tips and Best Practices

By David Keener

http://www.keenertech.com

Version 1.1 - April 28, 2012

Introduction

Ruby on Rails is an exciting technology…a well-crafted framework with innovative philosophies baked in to facilitate Agile Web Development

But even so, there are pitfalls…

A few simple tips can help you avoid the most common pitfalls

The Log File is Your Friend

You can’t optimize if you don’t know what your code is doing -- leverage the log file….

- Beginning Rails developers write inefficient code
  - Log File: /log/development.log
- Shows all web page parameters
  - Posted Fields
  - URL Parameters
- Shows all executed SQL

A Sample Log File

```
Processing LoginController#index (for 127.0.0.1 at 2009-08-27 03:33:44) [POST]
Parameters: {"x"=>"25", "y"=>"10", "authenticity_token"=>"bVOEM1qk1F4AH0=",
    "login_name"=>"dkeener@keenertech.com", "password"=>"sample"}
[4;36;1mUser Columns (2.5ms)[0m   [0;1mSHOW FIELDS FROM `users`[0m
 [4;35;1mUser Load (40.8ms)[0m   [0mSELECT * FROM `users` WHERE
    (`users`.`password` = 'gHyfrds76jD' AND `users`.`email` =
    'dkeener@keenertech.com') LIMIT 1[0m
[4;36;1mProfile Columns (2.1ms)[0m   [0;1mSHOW FIELDS FROM `profiles`[0m
[4;35;1mProfile Load (1.2ms)[0m   [0mSELECT * FROM `profiles` WHERE
    (`profiles`.`user_id` = 3) LIMIT 1[0m
[4;36;1mUser Update (0.3ms)[0m   [0;1mUPDATE `users` SET `login_count` = 4,
    `updated_at` = '2009-08-27 07:33:45' WHERE `id` = 3[0m
Redirected to http://127.0.0.1:3000/
Completed in 624ms (DB: 48) | 302 Found [http://127.0.0.1/login]
```

My Program Blew Up on a Query

"I was doing a `find(45)` and my code blew up!"

- `find(#)` raises a RecordNotFound exception if it doesn't find a matching row
- But `find_by_id(#)` returns nil if not found
- And the `first`, `all` and `last` methods return nil if they fail

It's easy to forget about this…

Column Definitions

Rails makes building your database easy…

```
rails generate model User first_name:string
    last_name:string login_name:string
    password:string
```

- Defaults to NULL-able columns
  - If it’s required, it should be NOT NULL
- Strings => varchar(255)
  - Why worry about storage? Just use the default size
  - Let model validations handle size constraints (if any)

A Typical Migration (Excerpt)

```
def self.up
  create_table :users do |t|
    t.string    :first_name, :null => false
    t.string    :last_name, :null => false
    t.string    :login_name, :null => false
    t.string    :email, :null => false
    t.string    :password
    t.integer   :login_count, :null => false, :default => 0
    t.boolean   :is_active, :null => false, :default => true
    t.timestamps
  end
end
```

Foreign Key Constraints

Do you use foreign key constraints or not?

- Rails discourages foreign keys
- Rails promotes enforcement of data integrity via the model (with validations)
- Rails defines model relationships (with associations)

Do you need foreign keys?

My answer: It depends….

Think of your database
as a source of water.

If your app is a…

…**Walled Fortress**, with a well in the central courtyard that can only be accessed through controlled means – then you don’t need foreign keys

…**Wildlife Preserve**, with a pond that serves as a watering hole for all sorts of animals – then you need foreign keys

Foreign Key Helper Code

```
module MigrationHelpers
  def fk(from_table, from_column, to_table)
    execute “alter table #{from_table}
                add constraint #{constraint(from_table, from_column)}
                foreign key (#{from_column}) references #{to_table}(id)”
  end

  def drop_fk(from_table, from_column)
    execute “alter table #{from_table}
                drop foreign key #{constraint(from_table, from_column)}”
  end

  def constraint(table, column)
    "fk_#{table}_#{column}"
  end
end
```

Conditional Logic in Migrations

Migrations are Ruby. You can do anything in Ruby...

- Find out what database is in use:

```
Rails 2.3.x
adapter = User.connection.instance_variable_get("@config")[:adapter]
Rails 3.x
adapter = connection.adapter_name.downcase.to_sym
```

- Find out what environment is in use:

```
if RAILS_ENV == ‘production’ …      # Rails 2.3.x
if Rails.env == :production …       # Rails 3.x
```

Especially useful if environments are different, e.g. – your laptop dev environment uses MySQL, but production uses Oracle

Fixture Recommendations

Fixtures are nice, but problematic.

- Can use fixtures for “lookup” data – e.g. states, countries, etc.
- Reference in both migrations and tests
- Also consider seeds for data
- For dynamic test data, use tools like FactoryGirl or Machinist

Loading a Fixture in a Migration

```
require 'active_record/fixtures'

class CreateCountries < ActiveRecord::Migration
  def self.up
    create_table :countries do |t|
    end

    Fixtures.create_fixtures('test/fixtures', File.basename("countries.yml", '.*'))
  end

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

Delegation

- Old-style convenience method to pull data from other models

```
# In the User model…
def birthday
    self.profile.birthday
 end
```

Does user have a profile?
Is caller aware of DB call?

- New-style using delegation

```
# In the User model…
delegate :birthday, :to => :profile
```

Limit what you bring back with `find`

- Use the will_paginate gem
- Or use limits: User.limit(5) # Rails 3.x

User.all(:limit => 5) # Rails 2.3.

Don’t create methods with the same name as associations – BAD IDEA

Conclusion

- Harness the power of Rails
  - But understand what it's doing for you
- Minimize database calls, but don't go crazy
  - Try eager loading in your `find` statements
- The console is your friend…use it…
- Consider data integrity in your apps
  - Or somebody will pay the price later
- Test, test, test
  - Try RSpec, Factory Girl and SimpleCov, etc.