Rails Migrations

Obligatory Slide

Eric Snyder

Use Ruby To

- Define and make changes to your database schema.
- Migrate data.
- Do whatever you need to do to migrate from point A to point B.
  - Manipulate files (YAML, XML)
  - ?
- Manage the migration process.

Benefits

- Synchronize database schema (& possibly data) among a team of developers.
- Easily update schema on release.
- Easily downgrade schema on rollback.
- Migrations are database agnostic (if you stay within the lines).
- Rails integration – rake tasks, generator

Supported / Not Supported

- Supported
  - MySQL
  - Oracle
  - PostgreSQL
  - SQLite
  - SQLServer
  - Sybase
- Not Supported
  - OpenBase

Creating A Migration

script/generate migration *migration_name*

- *migration_name* can be CamelCase or under_score
- hollow migration is created in db/migrate/nnn_migration_name (nnn is the next number in the sequence of migrations)

```
class Foo < ActiveRecord::Migration
  def self.up
  end

  def self.down
  end
end
```

Example Migration

```
class AddSsl < ActiveRecord::Migration
  def self.up
    add_column :accounts, :ssl_enabled, :boolean,
      :default => 1
  end

  def self.down
    remove_column :accounts, :ssl_enabled
  end
end
```

Another Example

```
class AddSystemSettings < ActiveRecord::Migration
  def self.up
    create_table :system_settings do |t|
      t.column :name,     :string
      t.column :label,    :string
      t.column :value,    :text
      t.column :type,     :string
      t.column :position, :integer
    end

    SystemSetting.create :name => "notice",
      :label => "Use notice?", :value => 1
  end

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

Going Outside The Lines

```
class AddSystemSettings < ActiveRecord::Migration
  def self.up
    execute “TRUNCATE TABLE foo”
  end

  def self.down

  end
end
```

Using A Model After Changing Its Table

```
def self.up
  add_column :beer_styles, :bjcp_category,
    :string

  BeerStyle.reset_column_information

  BeerStyle.find(:all).each do |style|
    style.bjcp_category = “dunno”
  end
end
```

Using A Model After Changing Its Table

- calls undef_method for each read method
- sets various hashes of column metadata to nil
- column metadata will be reloaded on next request

Transformations – Out Of The Box

- create_table
- drop_table
- rename_table
- add_column
- remove_column
- change_column
- add_index
- remove_index

Transformations – Out Of The Box

create_table(name, options) {} - notable options

- :primary_key (String) – PK column, defaults to “id”
- :id (boolean) – when false, no “id” column is generated, useful for habtm join tables
- :temporary (boolean) – when true creates a temp table
- :force – when true, force the creation of the table even if it already exists, always attempts to drop the table and ignores any failure to drop

Transformations – Out Of The Box

- column types - integer, float, datetime, timestamp, time, text, string, binary, boolean
- Mapping to native types , e.g. ActiveRecord::ConnectionAdapters::MySqlAdapter

```
def native_database_types #:nodoc
  { :primary_key => "int(11) DEFAULT NULL auto_increment PRIMARY KEY",
    :string     => { :name => "varchar", :limit => 255 },
    :text       => { :name => "text" },
    :integer    => { :name => "int", :limit => 11 },
    :float      => { :name => "float" },
    :datetime => { :name => "datetime" },
    :timestamp   => { :name => "datetime" },
    :time       => { :name => "time" },
    :date       => { :name => "date" },
    :binary     => { :name => "blob" },
    :boolean  => { :name => "tinyint", :limit => 1 }
  }
end
```

- column options - :limit, :default, :null

Saying It Nicely

- announce “Updating data”

== AddMyTable: Updating data ==========

- say “Updating data”

-- Updating data

- say_with_time { say “oooooh” }

-- oooooh

-> 0.0001s

Migrate To Migrations

- rake db:schema:dump - creates db/schema.rb
- put the contents of schema.rb into a migration up method.

schema.rb

- Created/re-created during execution of migrations.
- rake db:schema:dump
- rake db:schema:load

Transactional Migrations

- Migrations are not atomic by default.
- Migrations cannot be isolated.
- You can wrap a migration in a transaction (if your DBMS supports transactional DDL)
- There is a plugin to make transactional DDL the default.
  - svn://rubyforge.org//var/svn/redhillonrails/trunk/vendor/plugins/transactional_migrations

Quirks

- migration filename and class name are related. e.g. 001_foo – class must be named Foo or you will get “uninitialized constant error”

ActiveRecord::IrreversableMigration

```
def self.up
  Tag.find(:all).each { |tag| tag.destroy if
    tag.pages.empty? }
end

def self.down
  # Not much we can do to restore deleted data
  raise IrreversibleMigration
end
```

Migrations & Capistrano

- install capistrano - `sudo gem install capistrano`
- 'capistranoize' your app - `cap -A` *`rails_project_dir`*
- fiddle with config/deploy.rb
- rake remote:deploy_with_migrations
  - update code (svn update)
  - migrate
  - update symlink

Foreign Key Support

- The controversy
- foreign_key_schema plugin
  - adds foreign key support to ActiveRecord::SchemaDumper – schema.rb will contain add_foreign_key_constraint calls
  - adds add_foreign_key_constraint and remove_foreign_key_constraint to ActiveRecord::ConnectionAdapters::SchemaStatements – mixed into ActiveRecord::ConnectionAdapters::AbstractAdapter
  - MySQL & PostgreSQL only

Do It

- rake db:migrate
- rake db:migrate VERSION=x

Our Experiences

- What we did before.
- The move to migrations.
- What we have now.

Discussion