Beginning Ruby on Rails

Session Two

Models

“I don’t wake up for less than $10,000 a day.”

Models in Rails

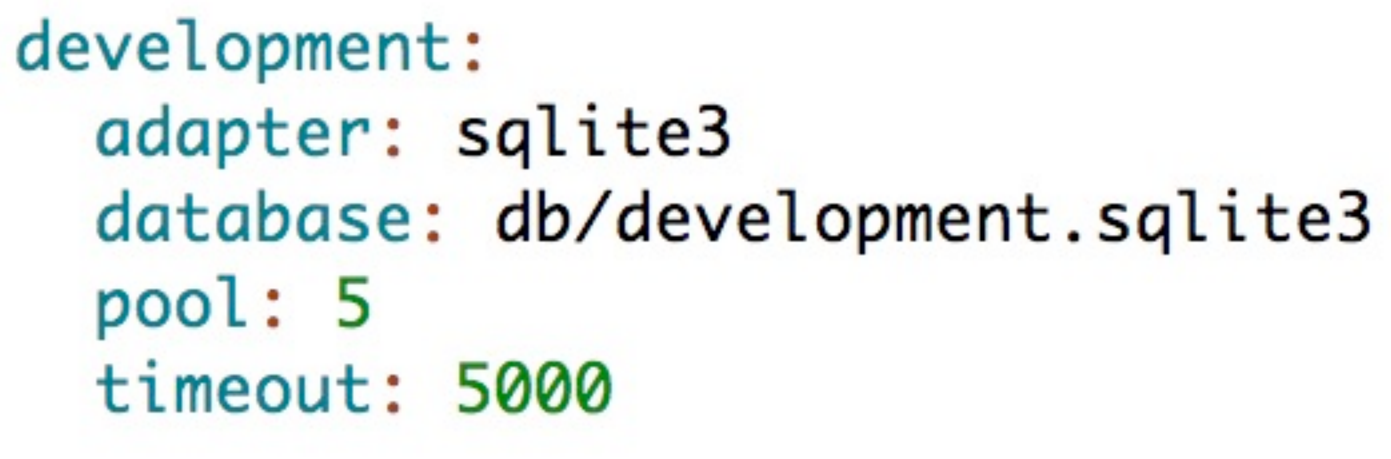
- Represent the data in the application and the rules to manipulate that data.
- Manage interaction with a corresponding database table.
- The bulk of your application’s business logic will be concentrated in the models.

Supported Databases

- MySQL
- Oracle
- PostgreSQL
- SQL Server
- SQLite
- Sybase

SQLite3 Config

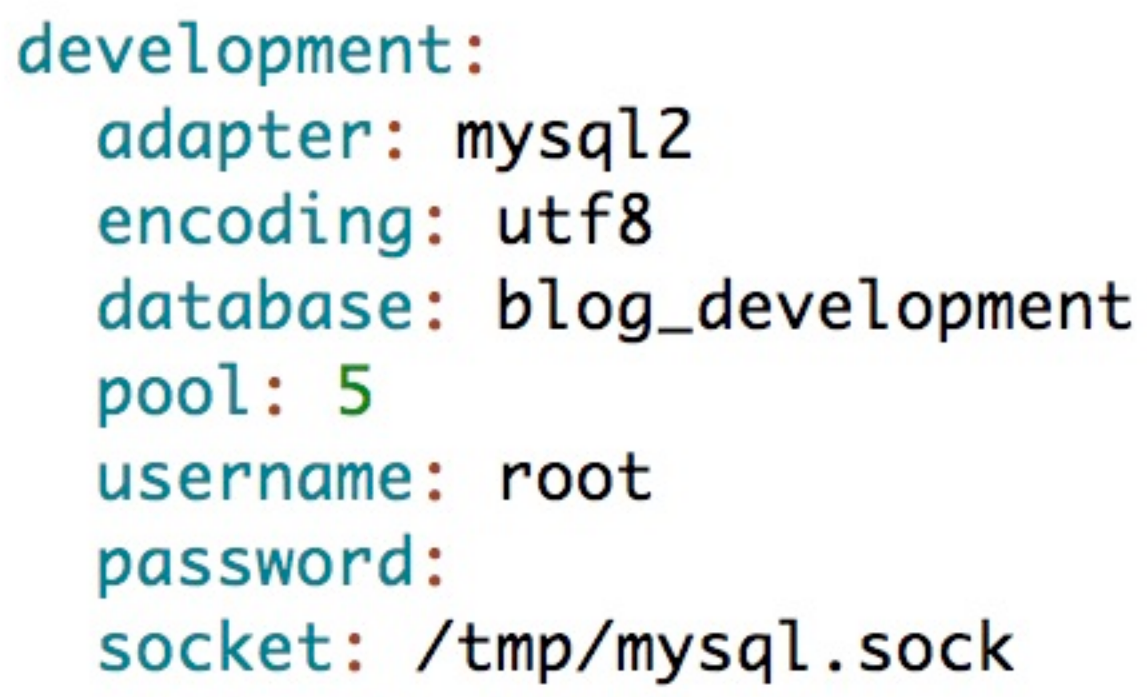
- Edit the file config/database.yml



```
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000
```

MySQL Config

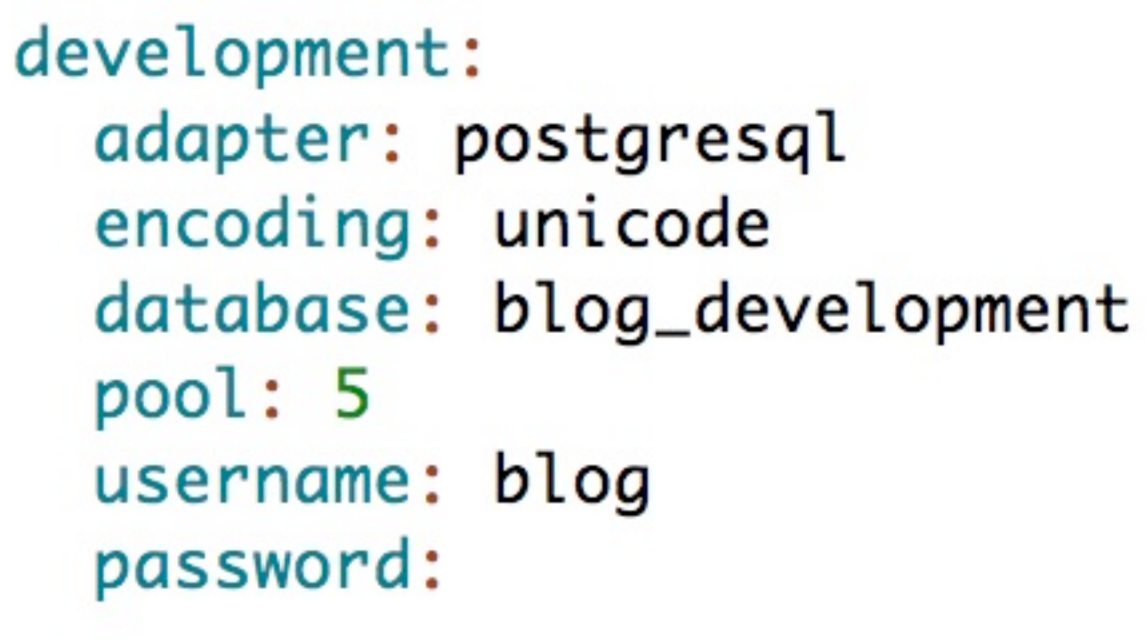
- Edit the file config/database.yml



```
development:
  adapter: mysql2
  encoding: utf8
  database: blog_development
  pool: 5
  username: root
  password:
  socket: /tmp/mysql.sock
```

PostgreSQL Config

- Edit the file config/database.yml



```yaml
development:
  adapter: postgresql
  encoding: unicode
  database: blog_development
  pool: 5
  username: blog
  password:
```

The Posts Model

- Edit the file app/models/post.rb
  - Not much to see here...

```
class Post < ActiveRecord::Base
end
```

Active Record

- An implementation of the object-relational mapping (ORM) pattern described by Martin Fowler.
- Automated mapping between classes and tables, attributes and columns.
- Direct Manipulation.

Rails Console

- The console is IRB with Rails preloaded
- Go to your blogs directory
  - Type “rails console”
  - Type “exit” to get out

CRUD

- Create
- Read
- Update
- Delete

Create

- Post.create :title => “First Post”

- p = Post.new
- p.title = “Second Post”
- p.save

Read

- p = Post.all
- p = Post.first
- p = Post.last

- p = Post.find 2
- p = Post.find_by_title “First Post”

Update

- p = Post.find 2
- p.title = “2nd Post”
- p.save

- p = Post.find 2
- p.update_attributes :title => “2nd Post”

Delete

- p = Post.find 2
- p.destroy

- p = Post.find 2
- p.delete

More Active Record

- Query Conditions:
  - order, limit, offset, group, having
- Calculations:
  - count, average, minimum, maximum

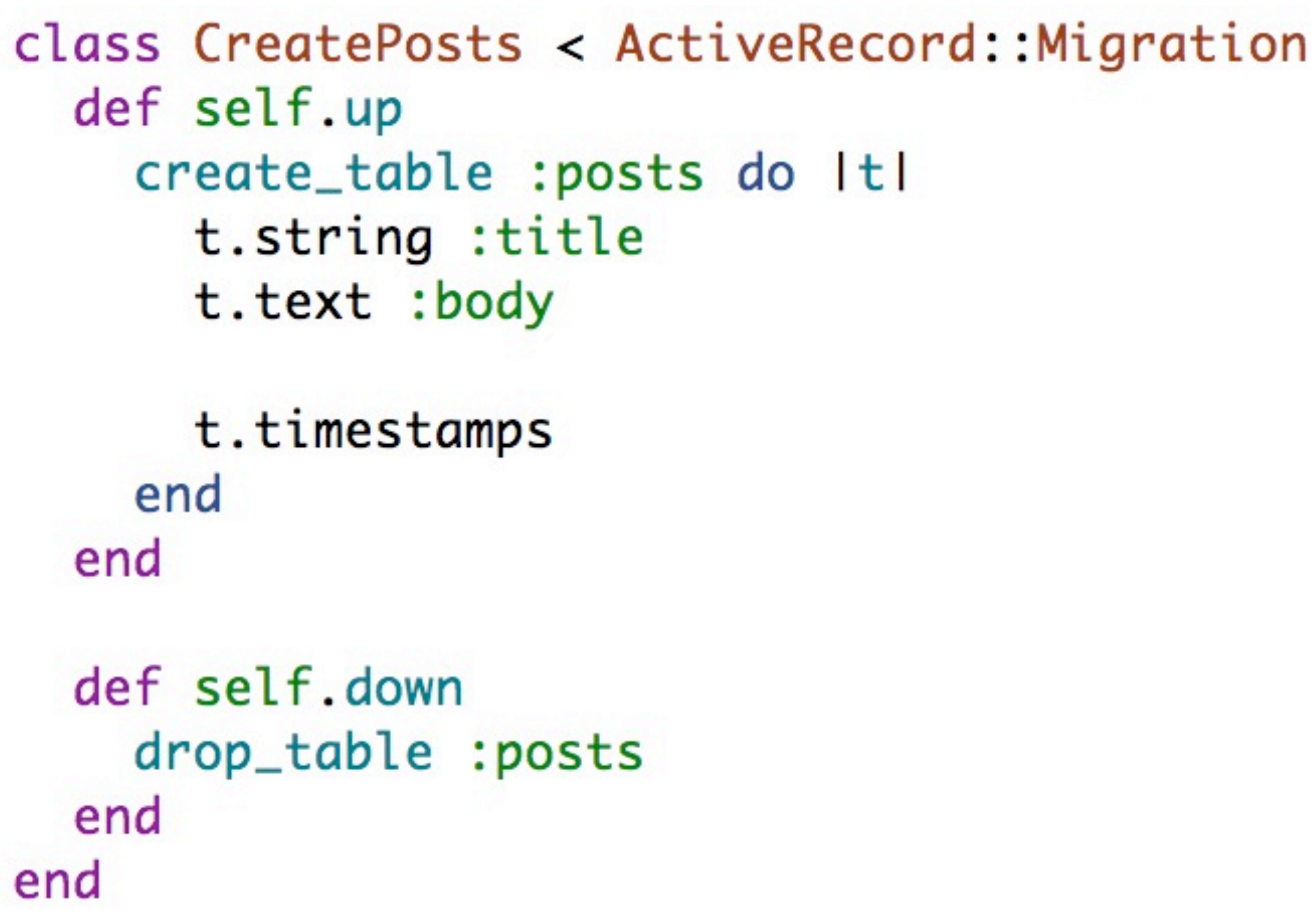
More Examples

- n = Post.count
- d = Post.maximum :created_at
- p = Post.order “created_at DESC”
- p = Post.order(“title ASC”).limit(2)

Migrations

Post Migration

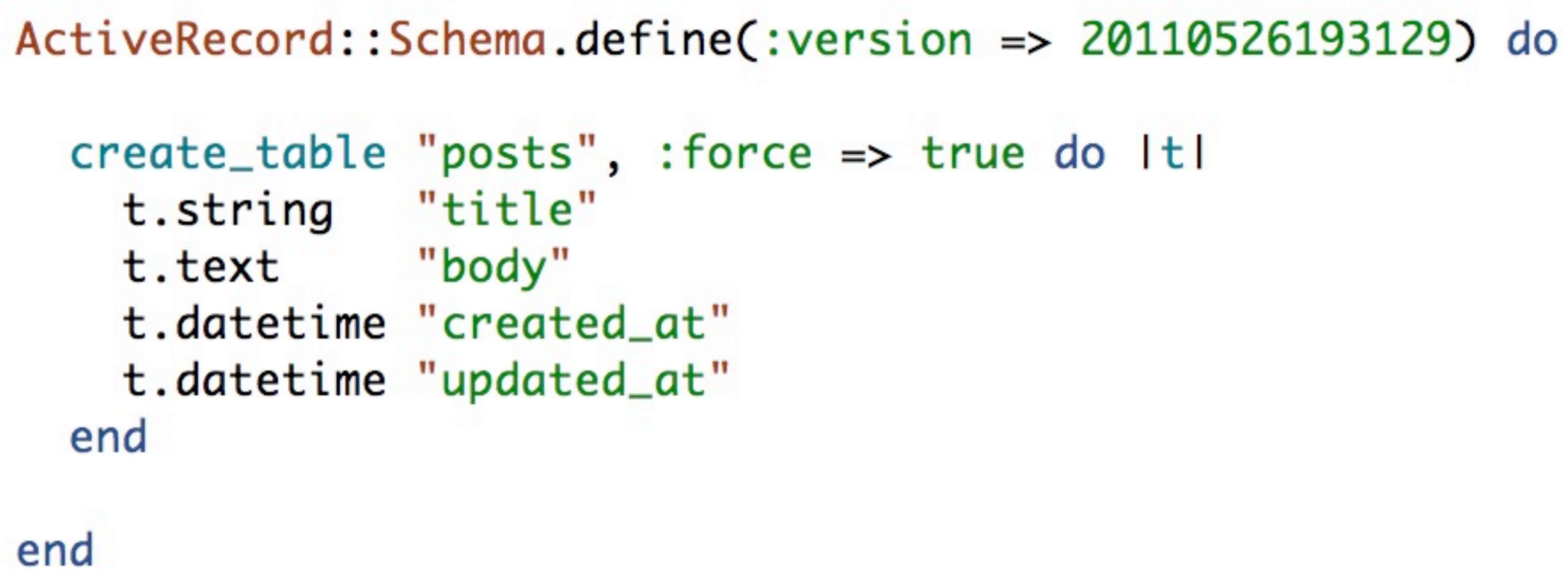
- Edit the file db/migrate/*_create_posts.rb



```ruby
class CreatePosts < ActiveRecord::Migration
  def self.up
    create_table :posts do |t|
      t.string :title
      t.text :body

      t.timestamps
    end
  end

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

The Schema

- Edit the file db/schema.rb



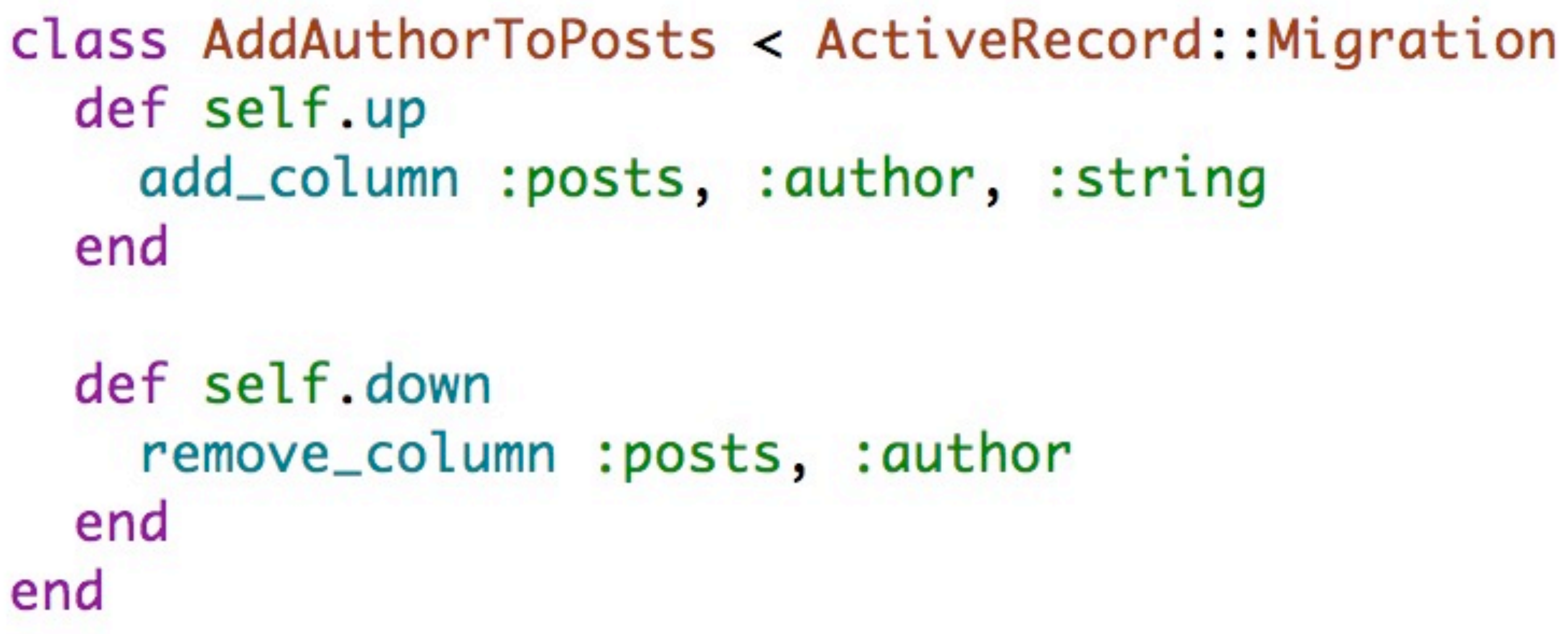
```ruby
ActiveRecord::Schema.define(:version => 20110526193129) do

  create_table "posts", :force => true do |t|
    t.string   "title"
    t.text     "body"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

end
```

Add a Column

- Posts need authors
    - rails generate migration add_author_to_posts author:string
    - rake db:migrate

Author Migration

- Edit db/migrate/*_add_author_to_posts.rb



```ruby
class AddAuthorToPosts < ActiveRecord::Migration
  def self.up
    add_column :posts, :author, :string
  end

  def self.down
    remove_column :posts, :author
  end
end
```

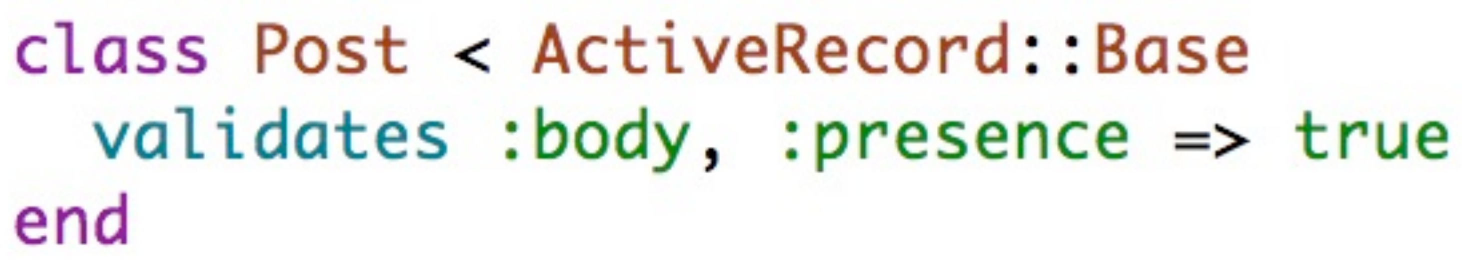
Validations

Protect Your Data

- Remember: models represent rules for manipulating the application data.
- Validation ensures that only good data makes it into the database.

No Empty Posts

- Edit app/models/post.rb



```ruby
class Post < ActiveRecord::Base
  validates :body, :presence => true
end
```

Common Validations

- :uniqueness
- :length
    - :minimum, :maximum, :within
- :inclusion, :exclusion
    - :in
- :numericality

Strict Titles

- Edit app/models/post.rb

```
class Post < ActiveRecord::Base
  validates :body, :presence => true
  validates :title, :presence => true,
    :uniqueness => true,
    :length => { :within => 1..20 },
    :exclusion => { :in => ["Title", "Post"] }
end
```

Testing Data

- Validations automatically run before data is saved to the database.
- Test for manually with the valid? method:
  - p = Post.new
  - p.valid?
  - p.errors

Associations

Let's Add Comments

- Posts and Comments are associated
  - Each Post *has many* Comments
  - Each Comment *belongs to* a Post

Generate the Model

- rails generate model Comment author:string body: text post:references
- rake db:migrate

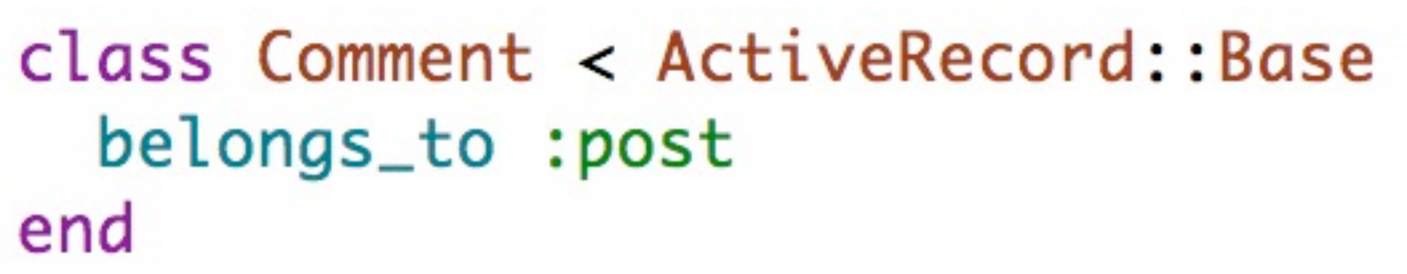
Post Associations

- Edit app/model/post.rb

```
class Post < ActiveRecord::Base
  has_many :comments
end
```

Comment Associations

- Edit app/model/comment.rb



```ruby
class Comment < ActiveRecord::Base
  belongs_to :post
end
```

Other Associations

- has_one
- has_many :through
- has_one :through
- has_and_belongs_to_many

belongs_to Methods

- post
- post=
- build_post
- create_post

has_many Methods

- comments
- comments<<
- comments.delete
- comments=
- comment_ids
- comment_ids=
- comments.clear
- comments.empty?
- comments.size
- comments.find
- comments.exists?
- comments.build
- comments.create

Using Associations

- p = Post.first
- p.comments
- p.comments.create :author => ”Tony”, :body => ”Test comment”
- p.comments.size

Using Associations

- c = Comment.new :post => p
- c.author = “Tony”
- c.body = “Another comment”
- c.save