Introduction to Active Record

A talk on our Object-relational Mapper

by Hunter T. Chapman

Introduction to ActiveRecord

- Just an introduction -- Mostly High Level
- What are the parts and how do they work
- Mini how-to
- This is gonna be awesome

Awesome Docs!

http://edgeguides.rubyonrails.org/active_record_migrations.html

The ActiveRecord Docs are some of the best you will find. Thorough, clean, simple. Lots of sample code to reference. Moral of the story == Use the Docs

ActiveRecord: What is it?

- Active Record is a gem (primarily) used in Rails for interacting with databases using object-relational mapping (ORM)

- ORM == creating persistent database objects that you can easily manipulate as Ruby objects.

ActiveRecord

- Get comfy, you will be using this… A LOT
- NBD – Turns out AR is super easy.

The Broad Strokes

Naming Conventions

- Class names singular, CamelCase
- Table names plural, snake_case

Names that ActiveRecord Can Match	
class name	*table name*
User	*users*
LineItem	*line_items*
Deer	*deer*
Person	*people*

Naming Conventions

- Class names singular, CamelCase
- Foreign key names singular, snake_case

Names that ActiveRecord Can Match	
class name	*foreign_key*
User	*user_id*
LineItem	*line_item_id*
Deer	*deer_id*
Person	*person_id*

Pro Tip:

Singular, Plural, snake_case, CamelCase

This stuff matters. You will spend hours tracking down issues with your schema only to find you missed an “s” on the end of one line that you’ve looked at a hundred times already. Super Not Cool!

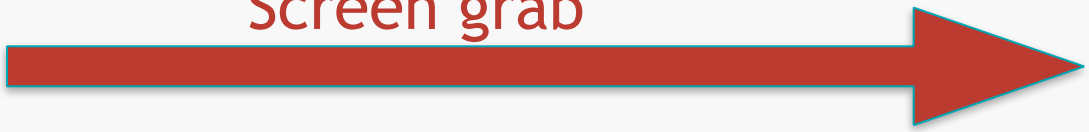
Active Record Sugar

Groovy Stuff AR does for you!

- Managing tables
- Mapping Ruby classes to database tables
- Associations between classes
- Validations

Active Record is BIG

ActiveRecord Source
Screen grab



You can, and will learn
something new about
AR every time you use
it for the foreseeable future.

For most of you this is the LARGEST
code base you have seen so far.

AR: What are the parts?

- Migrations – Used to build tables, a blueprint how our DB will represent objects.
- Models – Apply universal methods to table objects. Relationships and Validations live here.
- Control Code – Actual interface code for DB.

Active Record Parts

```
### Migrations ###
class CreatePersons < ActiveRecord::Migration
create_table "persons" do |t|
  t.string :first_name
  t.integer :age
end

### Models ###
class Person < ActiveRecord::Base
  has_many :dogs
  belongs_to :corperate_overlord
end

### Control ###
tom = Person.new
tom.first_name = "Tom"
tom.age = 28
tom.save
tom.dogs.create(name: "Fluffy the Fearless")
```

Basic ActiveRecord Workflow

1. Write Migrations
   - Blueprint of your DB
2. Run All Migrations
   - Once per change in DB schema
3. Build Models
   - Establish ORM Connection & Set up relationships
4. Implement control code
   - Manipulate objects and persist changes to DB

Inherit from ActiveRecord

- Two primary classes in ActiveRecord module

```
module ActiveRecord
  class Migration
    # ...
  end

  class Base
    # ...
  end

  # ...
end
```

Migrations

ActiveRecord Migrations
File naming conventions

20141020120711_create_users.rb

Key Points:

Timestamp: This tells the app the order to run migrations.

Action word: Describes the primary action of the migration.

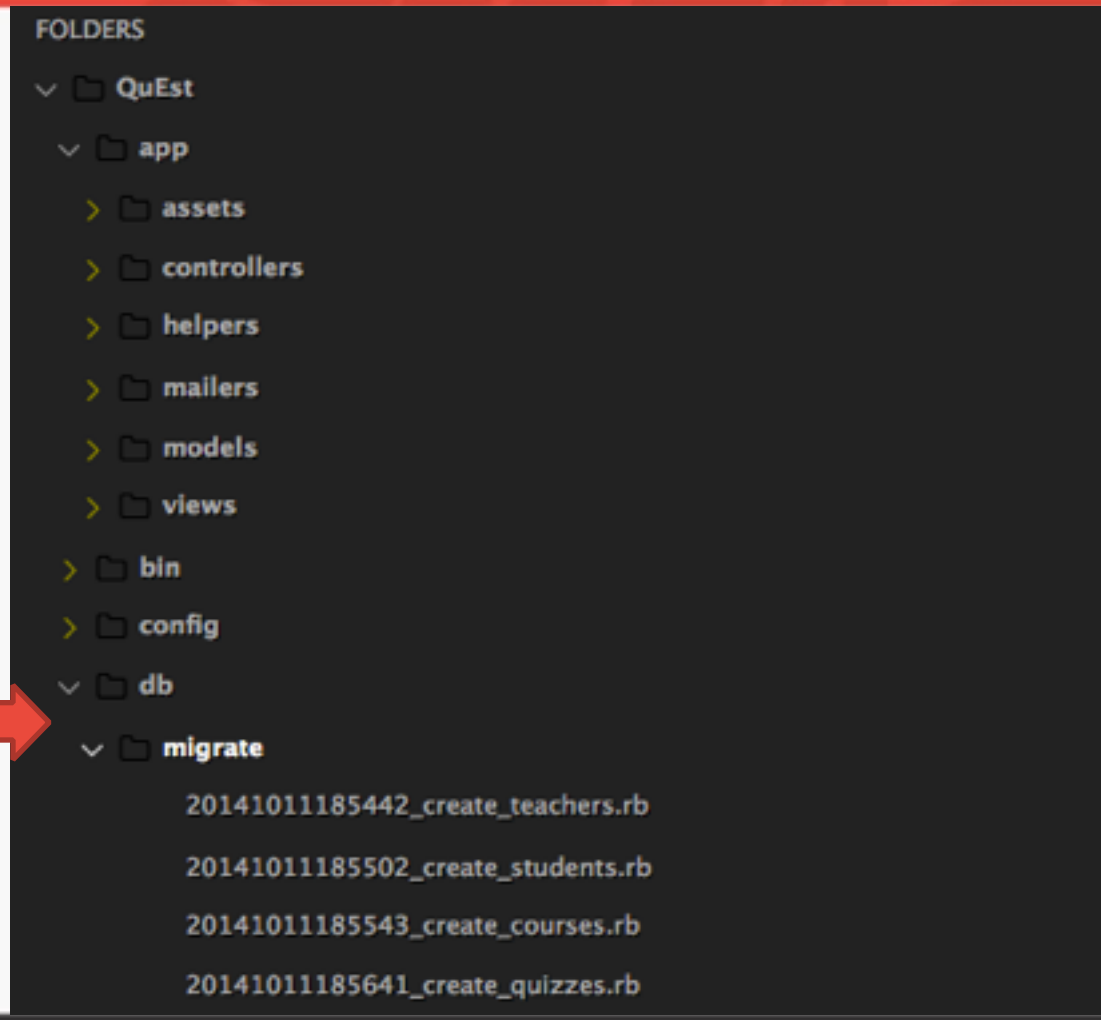
Table word: Describes the table effected by the migration.

Conventions:

snake_case, Action name is singular, Table name is plural.

ActiveRecord Migrations
File Location

Migrations live in:
ROOT/db/migrate



ActiveRecord Migrations

- Build database schema by writing migrations

```
# root/db/migrations/201407102947_create_oranges.rb

class CreateOranges < ActiveRecord::Migration
  def change
    # what do we want to do with the database
  end
end
```

ActiveRecord Migrations

- Add code to the change method to build out a table

```
class CreateOranges < ActiveRecord::Migration
  def change
    create_table :oranges do |t|
      t.integer :diameter
      t.string  :name
    end
  end
end
```

ActiveRecord Migrations

- Alter database schema by writing and running migrations

```
class AddOrangeTreeIdToOranges < ActiveRecord::Migration
  def change
    add_column :oranges, :orange_tree_id, :integer
    remove_column :oranges, :name
  end
end
```

Creating Tables with Migrations

orange_trees
id
age
height
created_at
updated_at

oranges
id
diameter
orange_tree_id
created_at
updated_at

Creating Tables with SQL

```
CREATE TABLE orange_trees (
 id INTEGER PRIMARY KEY
    AUTOINCREMENT,
 age INTEGER,
 height INTEGER,
 created_at DATETIME,
 updated_at DATETIME);
```

orange_trees
id
age
height
created_at
updated_at

Creating Tables with Migrations

```
class CreateOrangeTrees < ActiveRecord::Migration
  def change
    create_table :orange_trees do |t|
      t.integer :age
      t.integer :height

      t.timestamps
    end
  end
end
```

orange_trees
id
age
height
created_at
updated_at

Creating Tables with SQL

```
CREATE TABLE oranges (
 id INTEGER PRIMARY KEY
    AUTOINCREMENT,
 diameter INTEGER,
 orange_tree_id INTEGER,
 created_at DATETIME,
 updated_at DATETIME,
 FOREIGN KEY(orange_tree_id)
 REFERENCES orange_trees);
```

oranges
id
diameter
orange_tree_id
created_at
updated_at

Creating Tables with Migrations

```
class CreateOranges < ActiveRecord::Migration
  def change
    create_table :oranges do |t|

      t.integer :diameter
      t.references :orange_tree
      t.timestamps

    end
  end
end
```

oranges
id
diameter
orange_tree_id
created_at
updated_at

Creating Foreign Keys

ActiveRecord gives us three ways to create a foreign key in a Migration.

You only need ONE of these to do the job.

```
class CreateOranges < ActiveRecord::Migration
  def change
    create_table :oranges do |t|
      t.integer :diameter
      t.integer :orange_tree_id
      t.belongs_to :orange_tree
      t.references :orange_tree
      t.timestamps
    end
  end
end
```

All the same

oranges
id
diameter
orange_tree_id
created_at
updated_at

Migration Methods Note:

def change is the thing you will use most often.

http://stackoverflow.com/questions/20890510/rails-migration-change-vs-up-down-methods

```
class AddOrangeTreeIdToOranges < ActiveRecord::Migration

  def change
  end

    –VS:–

  def self.up
  end

  def self.down
  end

end
```

Common Migration Datatypes

```
:boolean        :primary_key
:datetime       :string
:decimal        :text
:float          :time
:integer        :timestamp
```

Sample List only
Consult your docs for accurate data types based on the DB in use.

Models

ActiveRecord Models
File naming conventions

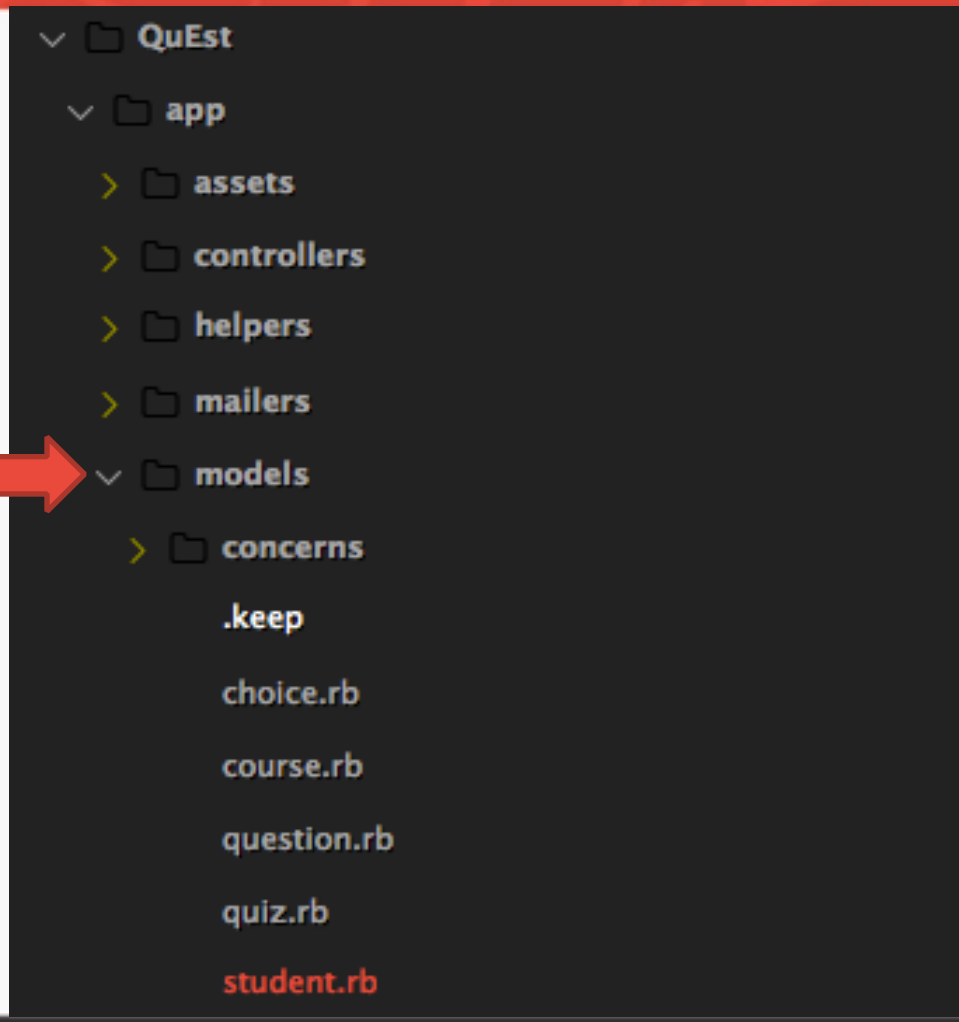
student.rb

Key Points:

- snake_case
- Singular
- Filename matches applicable table

ActiveRecord Models
File Location

Models live in:
ROOT/app/models



Models

ActiveRecord Models have 3 key functions:

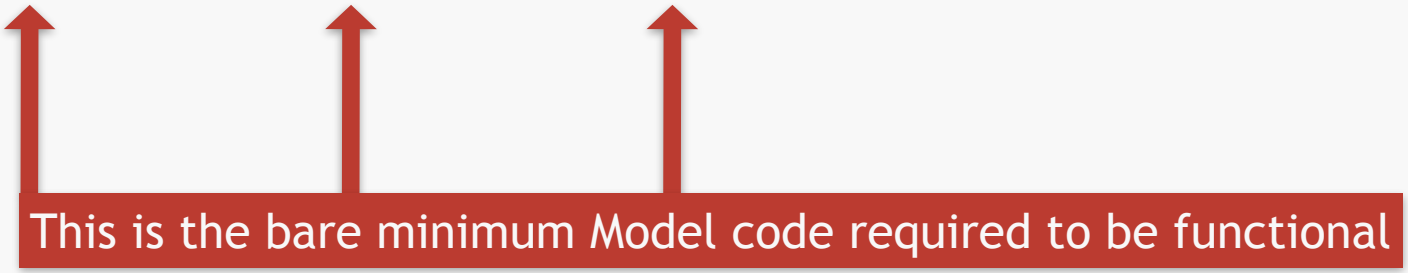
1. Connect DB & Ruby objects
2. Define Associations between classes
3. Define Validation & Callback events

Models Map Classes to Tables

```
class OrangeTree < ActiveRecord::Base
end
```

Models Map Classes to Tables

```
class OrangeTree < ActiveRecord::Base
end
```



Models Map Classes to Tables

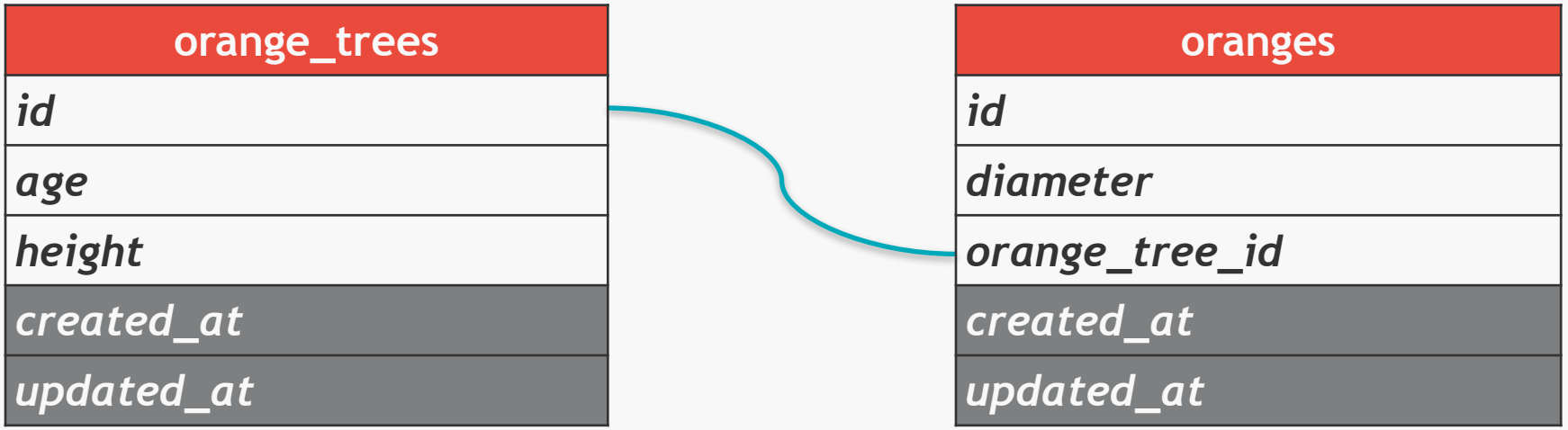
```
class OrangeTree < ActiveRecord::Base
end
```

Modeling State	
Ruby	*Database*
Classes	*Tables*
Instances of classes	*Rows*
Instance variables	*Fields*

Associations Between Classes

We are dealing with relational databases.

Relationships == Associations



Define Associations in a Model:

```
# root/app/models/orange_tree.rb

class OrangeTree < ActiveRecord::Base
  has_many   :oranges
end
```

```
# root/app/models/orange.rb

class Orange < ActiveRecord::Base
  belongs_to :orange_tree
end
```

Associations Between Classes

Models are the place we tell ActiveRecord how our classes are associated. Choose a method that will reflect the nature of those associations.

```
has_one :facebook

has_many :shows

has_many :student_classes
has_many :classes, through: :student_classes

belongs_to :employee
```

Associations Between Classes

oranges				
id	*diameter*	*orange_tree_id*	*created_at*	*updated_at*
1	***2***	***1***	***2014-03-22***	***2014-03-22***
2	***4***	***2***	***2014-03-22***	***2014-03-22***

```
orange = Orange.find(1)
# => #<Orange:0x003frd5b9t8a24 @id=1, @diameter=2 ...>

orange.orange_tree
# => #<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=5, @height=5 ...>
```

Associations Between Classes

oranges				
id	*diameter*	*orange_tree_id*	*created_at*	*updated_at*
1	***2***	***1***	***2014-03-22***	***2014-03-22***
2	***4***	***2***	***2014-03-22***	***2014-03-22***

```
tree = OrangeTree.find(1)
# => #<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=5, @height=5 ...>

tree.oranges
# => [#<Orange:0x003frd5b9t8a24 @id=1, @diameter=2 ...>]
```

Creating Associations Between Classes

oranges				
id	*diameter*	*orange_tree_id*	*created_at*	*updated_at*
1	***2***	***1***	***2014-03-22***	***2014-03-22***
2	***4***	***2***	***2014-03-22***	***2014-03-22***
3	***3***	***1***	***2014-03-22***	***2014-03-22***

```
tree = OrangeTree.find(1)
# => #<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=5, @height=5 ...>

tree.oranges
# => [#<Orange:0x003frd5b9t8a24 @id=1, @diameter=2 ...>]

tree.oranges.create(diameter: 3)
# => [#<Orange:0x0029ahaf089098 @id=2, @diameter=3 ...>]
```

Creating Associations Between Classes

oranges				
id	*diameter*	*orange_tree_id*	*created_at*	*updated_at*
1	***2***	***1***	***2014-03-22***	***2014-03-22***
2	***4***	***2***	***2014-03-22***	***2014-03-22***
3	***3***	***1***	***2014-03-22***	***2014-03-22***

```
tree = OrangeTree.find(1)
# => #<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=5, @height=5 ...>

orange = Orange.create(diameter: 5)

tree.oranges << orange
# => [#<Orange:0x003frd5b9t8a24 @id=4, @diameter=5 …>]
```

Same result as previous slide. IMHO: I prefer the previous way.

Foreign Keys DON'T go in Models

- So turns out you have to do more than just define relations in the Model. Luckily not much more…
- Two related tables are referenced at the DB level by Foreign Keys. These keys are created when you build a Migration.
- You must define a Foreign Key in a Migration and run those migrations before your Model code will function.
- Where you place the Foreign Key matters.
  (it goes on the table for the class declaring the belongs_to association)

Validations

- Basically, Validations prevent writing to the database if the data does not meet certain requirements.

Validations

- IMPORTANT— Always make sure your migrations and models are Rock Solid before implementing Validations.

Validations

```
CREATE TABLE orange_trees (
 id INTEGER PRIMARY KEY
    AUTOINCREMENT,
 age INTEGER NOT NULL,
 height INTEGER NOT NULL,
 created_at DATETIME,
 updated_at DATETIME);
```

Validations

```
CREATE TABLE orange_trees (
 id INTEGER PRIMARY KEY
    AUTOINCREMENT,
 age INTEGER NOT NULL,
 height INTEGER NOT NULL,
 created_at DATETIME,
 updated_at DATETIME);
```

```
class OrangeTree < ActiveRecord::Base
  validates :age, presence: true
  validates :height, presence: true
end
```

Validations

- ActiveRecord has built in validators like:
    - presence
    - format
    - uniqueness
    - etc.

Convention over Configuration

Lots of really smart people have already cranked through the vast majority of problems you are going to face. They were kind enough to hook you up with the fruits of their labor, or what we affectionately call Best Practices.

- Convention: Write less code, straight forward and clean approach to make your life easier.
- Configuration: Breaks from standard practice when absolutely required to get the job done.

Custom Validations

- You can also write your own validators

```
class Orange < ActiveRecord::Base
  validate :legit_diameter

  def legit_diameter
    errors.add_to_base("Not Legit") unless orange.diameter > 2
  end
end
```

Failed Validations add Errors

- Before writing to the database, ActiveRecord runs validations from the model

- If a validation fails:

  - An error is added to the object
  - ActiveRecord will not write to the database

Callbacks

Callbacks are methods that will run at a given point in your objects life cycle within the DB.

Callbacks

A few of the many callbacks available

before_validation
before_save
before_update
before_create
before_destroy

after_validation
after_create
after_save
after_update
after_destroy

Callbacks

```
class Orange < ActiveRecord::Base

  belongs_to :orange_tree

  after_initialize do |orange|
    puts “I made an Orange!!!”
  end

end
```

Control Code

Control Code

MODEL:

```
class OrangeTree < ActiveRecord::Base
    has_many :oranges
End
```

Control Code accesses this model:

```
tree = OrangeTree.new(age: 6, height: 15)
```

Now your program has access to this DB object in local memory

Mapping Classes to Tables

- Class methods:

  Retrieve records from table
- Instance methods:

  Read and write values

Class Methods

orange_trees				
id	*age*	*height*	*created_at*	*updated_at*
1	***5***	***5***	***2014-03-22***	***2014-03-22***
2	***6***	***6***	***2014-03-22***	***2014-03-22***

```
OrangeTree.all
# => [#<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=5, @height=5 ...>,
      #<OrangeTree:0x007fdd5b22b268 @id=2, @age=6, @height=6 ...>]

OrangeTree.find(1)
# => #<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=5, @height=5 ...>
```

Instance Methods Match Fields

orange_trees				
id	*age*	*height*	*created_at*	*updated_at*
1	***5***	***5***	***2014-03-22***	***2014-03-22***
2	***6***	***6***	***2014-03-22***	***2014-03-22***

```
tree = OrangeTree.find(1)
# => #<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=5, @height=5 ...>

tree.id       # => 1
tree.age      # => 5
tree.height   # => 5
```

Instance Methods Match Fields

orange_trees				
id	*age*	*height*	*created_at*	*updated_at*
1	***5***	***5***	***2014-03-22***	***2014-03-22***
2	***6***	***6***	***2014-03-22***	***2014-03-22***

```
tree = OrangeTree.find(1)
# => #<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=5, @height=5 ...>

tree.age = 9  # => 9
tree
# => #<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=9, @height=5 ...>
```

Instance Methods Match Fields

orange_trees				
id	*age*	*height*	*created_at*	*updated_at*
1	***9***	***5***	***2014-03-22***	***2014-03-22***
2	***6***	***6***	***2014-03-22***	***2014-03-22***

```
tree = OrangeTree.find(1)
# => #<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=5, @height=5 ...>

tree.age = 9  # => 9
tree
# => #<OrangeTree:0x007fdd5b9b4a20 @id=1, @age=9, @height=5 ...>
tree.save     # => true
```

CRUD

```
### C.R.U.D. ###

### CREATE ###
p1 = Person.create(first_name: "Tami", age: 26)
p2 = Person.new(first_name: "Bill", age: 46)
p2.save
### READ ###
p3 = Person.find(1)
p4 = Person.find_by_first_name("Bill")
### UPDATE ###
p1.update_attributes(age: 27)
### Destroy ###
p2.destroy
```

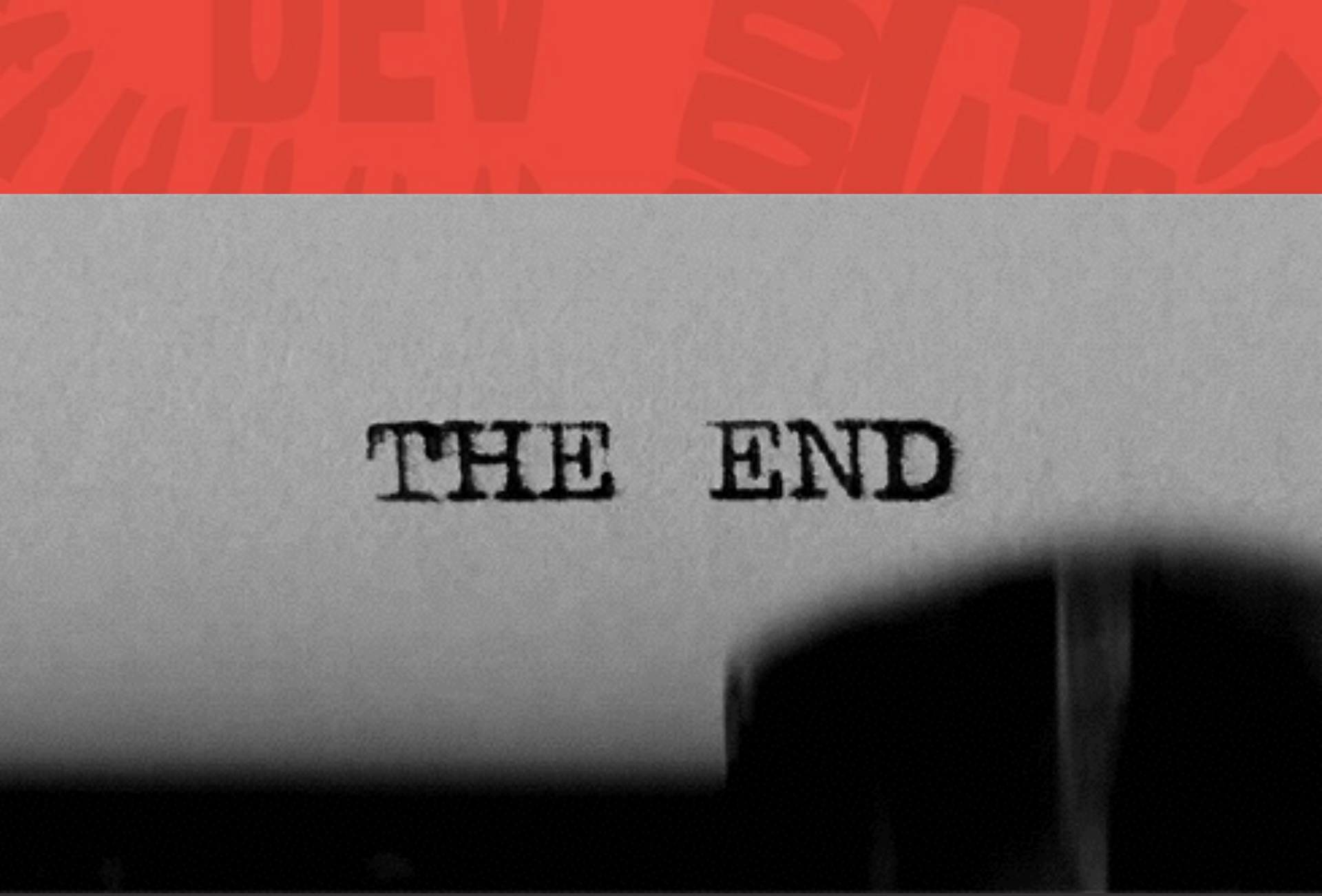
Parting Tip:
Use HIRB gem to clean up console

```
irb(main):007:0> require 'hirb'
=> true

irb(main):008:0> Hirb.enable
=> true

irb(main):009:0> User.all
D, [2015-09-22T09:21:44.791048 #37624] DEBUG -- :   User Load (0.5ms)  SELECT
"users".* FROM “users"

+——+————————+——-—+————————————————+————————————————-————————————+
| id | first_name | age |      created_at          |          updated_at          |
+——+——-——-——+——-—+————————————————+————————————-————————————+
| 1  | Hunter      | 18   | 2015-09-22 16:21:13 UTC | 2015-09-22 16:21:13 UTC |
+——+————————+——-—+————————————————+————————————————-————————————+
```


THE END