



Ruby on Rails Short Course: Advanced Model Relations

William Sobel

UC Berkeley RAD Lab

Outline of the day

1. Web apps, MVC, SQL, Hello World
2. Just enough Ruby
3. Basic Rails

Lunch break

4. **Advanced model relations**
5. AJAX & intro to testing
6. Configure and Deploy

Informal discussion: RoR and pedagogy

Section 4

- Review
 - Conventions
- Associations
 - One-to-one, one-to-many, many-to-many
- Transactions
- Advanced
 - Counters, Locking, Single-Table-Inheritance,
...

The Model

- Place all database access in the model
- Place all validations in the model
 - Do not validate values in the controller!
- Place all business logic in the model
 - All computations
 - All object relations
- Keep your view and controller clean

Be Conventional

- Name your tables the plural of the class
- Name your foreign key columns: <class>_id
- Join tables are named with the first table name _ second table name (alphabetical order) ex. accounts_users
- Use Migrations
 - Cross database SQL generation
 - Automatic Schema Management

Associations

- Powerful Meta-Programming Tools to Express Relationships Between Classes
- Supports
 - One-to-one (has and belongs to)
 - One-to-many
 - Many-to-many (using join table) habtm
 - Many-to-many (using join-model) hmt

A simple model, let's start with the DB

```
class CreateAuthors < ActiveRecord::Migration
  def self.up
    create_table :authors do |t|
      t.column :name, :string
      t.column :created_at, :datetime
    end

    create_table :books do |t|
      t.column :author_id, :integer
      t.column :title, :string, :limit => 64
      t.column :isbn, :decimal, :precision => 11
      t.column :sales, :decimal, :precision => 10, :scale => 2, :default => 0
    end

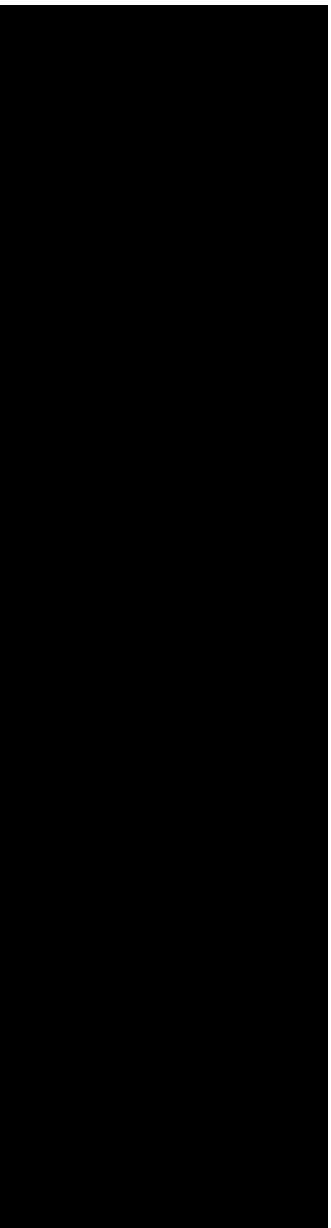
    def self.down
      drop_table :accounts
      drop_table :books
    end
  end
end
```

Associations

Now for the Active Record classes

```
class Author < ActiveRecord::Base
  has_many :books, :foreign_key => 'author_id', :class_name => 'Book'
end

class Book < ActiveRecord::Base
  belongs_to :author, :foreign_key => 'author_id', :class_name => 'Author'
end
```



The Power of Convention

```
class Author < ActiveRecord::Base
  has_many :books
end
```

```
class Book < ActiveRecord::Base
  belongs_to :author
end
```

```
>> a = Author.create(:name => "Dave Thomas")
❏ #<Author:0x34599f0 @attributes={"name"=>"Dave Thomas", "id"=>2, ...>>>
>> a.books.create(:title => "Programming Ruby", :isbn => "111111111")
=> #<Book:0x345047c @attributes={"isbn"=> "111111111", "sales"=>0,
    "title"=>"Programming Ruby", ...>
```

Has & Belongs to Many

Wait! That's Wrong! Books have more than one author

```
class Author < ActiveRecord::Base
  has_and_belongs_to_many :books
end
```

```
class Book < ActiveRecord::Base
  has_and_belongs_to_many :authors
end
```





Has & Belongs to Many

The Migration

```
class CreateHabtM < ActiveRecord::Migration
  def self.up
    create_table :authors_books, :id => false do |t|
      t.column :author_id, :integer
      t.column :book_id, :integer
    end

    # We no longer need this column
    remove_column :books, :author_id
  end

  def self.down
    drop_table :authors_books, :id => false
    add_column :books, :author_id, :integer
  end
end
```

Using Relations

Not as hard as you think...

```
dt = Author.create(:name => 'Dave Thomas')
ah = Author.create(:name => 'Andy Hunt')
b = Book.create(:title => 'Programming Ruby')

dt.books << b
ah.books << b

dt.books.map &:title
["Programming Ruby"]
ah.books.map &:title
["Programming Ruby"]

# Generates
SELECT * FROM books INNER JOIN authors_books ON books.id =
  authors_books.book_id WHERE (authors_books.author_id = 5 )
```

Including Relations

Optimize...

```
# Selects all the authors and pre-populates the books
# relation.
authors = Author.find(:all, :include => [:books])
```

```
# The SQL once again:
SELECT authors.`id` AS t0_r0, authors.`name` AS t0_r1,
authors.`created_at` AS t0_r2, books.`id` AS t1_r0,
books.`title` AS t1_r1, books.`isbn` AS t1_r2, books.`sales` AS
t1_r3 FROM authors LEFT OUTER JOIN authors_books ON
authors_books.author_id = authors.id LEFT OUTER JOIN books ON
books.id = authors_books.book_id
```

Beyond habtm

```
class Author < ActiveRecord::Base
  has_many :authors_books
  has_many :books, :through => :authors_books
end
```

```
class Book < ActiveRecord::Base
  has_many :authors_books
  has_many :authors, :through => :authors_books
end
```

```
class AuthorsBook < ActiveRecord::Base
  belongs_to :author
  belongs_to :book
end
```



The Migration

```
class CreateJoinModel < ActiveRecord::Migration
  def self.up
    create_table :authors_books2, :force => true do |t|
      t.column :author_id, :integer
      t.column :book_id, :integer
      t.column :royalty, :float
    end

    # Initialize the royalty column to 0.0
    ActiveRecord::Migration::execute("INSERT INTO authors_books2 (author_id, book_id,
    royalty) SELECT author_id, book_id, 0.0 FROM authors_books")


    drop_table :authors_books
    rename_table :authors_books2, :authors_books
  end

  def self.down
    ..
  end
end
```

Add some royalties to the join model

```
b = Book.find_by_title('Programming Ruby')
for ab in b.authors_books.find(:all, :include => [:author])
  ab.royalty = ab.author.name == 'Dave Thomas' ? 7 : 3
  ab.save
end
```

```
class Book < ActiveRecord::Base
  has_many :authors_books, :order => 'authors_books.royalty DESC'
  has_many :authors, :through => :authors_books,
    :order => 'authors_books.royalty DESC'
end
```

```
b = Book.find_by_title('Programming Ruby')
b.authors.map &:name
 ['Dave Thomas', 'Andy Hunt']
b.authors_books.map &:royalty
=> [7.0, 3.0]
```


Inheritance

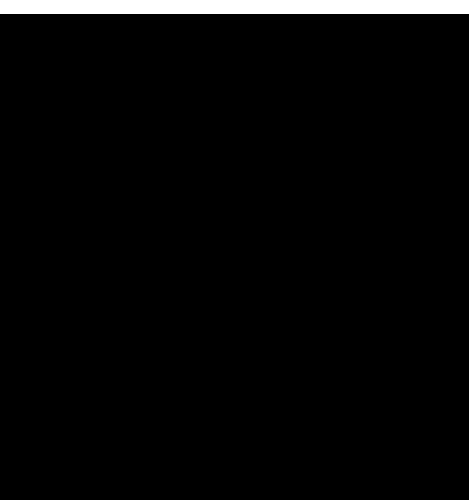
Rails supports Single-Table-Inheritance

```
class Person < ActiveRecord::Base
end
```

```
class Employee < Person
  belongs_to :manager
  belongs_to :department
end
```

```
class Manager < Employee
  has_many :employees
end
```

```
class Consultant < Person
  belongs_to :company
end
```



Rails supports Single-Table-Inheritance

```
bob = Employee.create(:name => "Bob")
mary = Manager.create(:name => "Mary")
tim = Consultant.create(:name => "Tim")
```

```
bob.manager = mary
bob.save
```

```
mary.employees.map &:name
#> ["Bob"]
```

```
jane = Manager.create(:name => 'Jane')
Manager.find(:all).map &:name
#> ['Mary', 'Jane']
```

```
jane.employees.map &:name
#> []
```

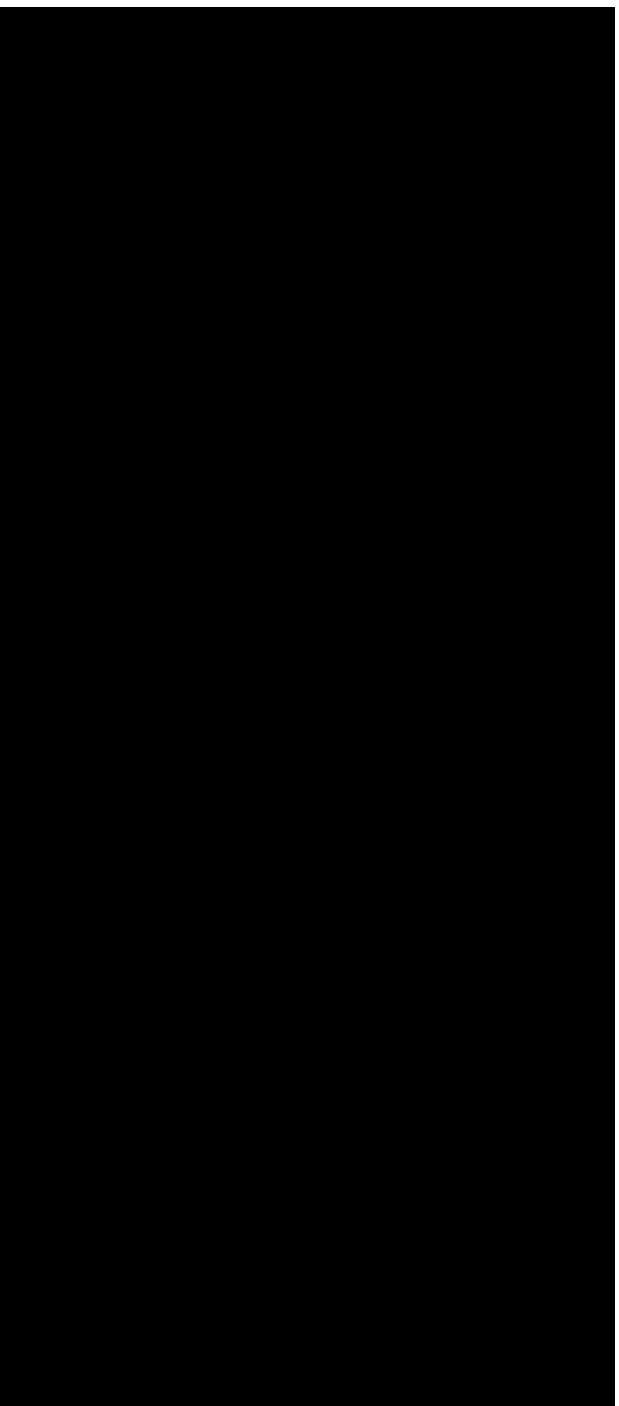
```
bob.manager = jane
jane.employees.map &:name
#> ['Bob']
```

```
Employee.find(:all).map &:name
#> ['Bob', 'Mary', 'Jane']
```

```
Person.find(:all).map &:name
['Bob', 'Mary', 'Tim', 'Jane']
```

Associate to any object

```
class Resource < ActiveRecord::Base
  belongs_to :resource, :polymorphic => true
end
```



Calculations

- Simple syntax for computed results
 - maximum
 - minimum
 - average
 - sum
 - count

Computed results

```
# Get the average price of all the books  
price = Book.average(:price)
```

```
# How many books?  
count = Book.count
```

```
:conditions - SQL where statement
```

```
:having - Having clause
```

Groups

```
prices = Book.maximum(:price,  
                        :group => 'Category')  
=> [[ 'Humor' , 7.95] , [ 'Fantasy' , 4.3] , ...]
```

Counting

```
num = Book.count [ 'price > ?' , price]  
  
num = Book.count [ 'price > ?' , price] ,  
  'left join authors_books on book_id =  
  books.id'  
  
num = Book.count_by_sql( '...' )
```

Transactions

```
Author.transaction(author1, author2) do
  # Do some work
  author1.books << book
  author2.books << book
end # Auto Commits

# If this transaction fails, author1 and
author2 are restored to their previous
state.
```

- Aggregations

```
class Money
  attr_accessor :amount, :currency

  def initialize(amount, currency)
    @amount = amount
    @currency = currency
  end

  def convert(currency)
    # Get rate
    @amount * rate
  end
end

class Book < ActiveRecord::Base
  composed_of :price, :class_name => Money,
    :mapping => [[:amount, :amount], [:currency, :currency]]
end
```


- Using Composed Object

```
price = Money.new(29.95, 'USD')  
book = Book.create(:title => 'Programming Ruby',  
                  :price => price)
```

```
book = Book.find_by_title('Programming Ruby')  
price = book.price
```

```
price.amount  
⇒ 29.95
```

```
price.convert('EUR')  
⇒ 22.02
```

Counter Caching

- Common pattern in web application
 - How many books does this author have?

```
add_column :authors, :authors_books_count,
           :integer, :default => 0
```

```
class AuthorsBook < ActiveRecord::Base
  belongs_to :author, :counter_cache => true
  ..
end
```

```
dt = Author.find_by_name('Dave Thomas')
b = Book.create(:title => 'Agile Rails 2')
dt.reload
dt.books << b
dt.authors_books_count
❏ 1
b2 = Book.find_by_title('Programming Ruby'); dt.books << b2
dt.reload
dt.authors_books_count
❏ 2
```

- Become a Tree...

```
create_table :groups do |t|  
  t.column :name, :string  
  t.column :parent_id, :integer  
end
```

```
class Group < ActiveRecord::Base  
  acts_as_tree :order => :name  
  # Creates parent and children relationships...  
end
```

```
root = Group.create(:name => 'root')  
root.children.create(:name => 'Child 1')  
root.children.create(:name => 'Child 2')
```

- Become a List...

```
create_table :thumbnails do |t|
  t.column :filename, :string
  t.column :picture_id, :integer
  t.column :position, :integer
end
```

```
class Picture < ...; has_many :thumbnails, :order => position; end
```

```
class Thumbnail < ActiveRecord::Base
  belongs_to :picture
  acts_as_list :scope => :picture
end
```

```
picture = Picture.new(:name => 'picture 1')
picture.thumbnails.create(:filename => 'f.gif')
picture.thumbnails.create(:filename => 'f.png')
picture.thumbnails.create(:filename => 'f.jpg')
```

```
picture.thumbnails[0].move_lower
picture.thumbnails[0].move_to_top
```

What's Next

- Not Every Model Persists
 - If it can CRUD, it can model...
 - Tie together RESTful apps. My Model ties to your REST interface.
 - No more WebService as separate service
- Alternatives to Relational
 - Ontology based data-stores
 - S3
 - SOLR, Ferret, ...
 - ?



Questions