ActiveRecord

Models and Migrations

Devpoint Labs - Jake Sorce / Dave Jungst

Making Models

Making a simple Person Model:

bundle exec rails g model person

```
  => invoke  active_record
        create    db/migrate/20150829183535_create_people.rb
        create    app/models/person.rb
```

Making a Person defining attributes Model:

bundle exec rails g model person name:string age:integer race:string hair_color:string alive:boolean

Migrations

Running Migrations:

```
bundle exec rake db:migrate
  => == 20150829183706 CreatePeople: migrating
==========================================
-- create_table(:people)
== 20150829183706 CreatePeople: migrated (0.0007s)
=================================
```

Rolling Back Migrations:

```
bundle exec rake db:rollback
```

Retrieving Records

Returning single records

Return a single record by ID:

```
Person.find(1)
  => #<Person id:1, first_name: ‘Jake’, last_name: ‘Sorce’>
```

Return the first record that matches

```
Person.find_by(first_name: ‘Dave’)
  => #<Person id:2, first_name: ‘Dave’, last_name: ‘Jungst’>
```

Retrieving Records (cont)

Return all records

```
Person.all
  => #<ActiveRecord::Relation [
        #<Person id: 1, first_name: ‘Jake’,..>,
        #<Person id: 2, first_name: ‘Dave’>
      ]>
```

Return all that match

```
Person.where(first_name: ‘Dave’)
  => #<ActiveRecord::Relation [
        <#Person id: 2, first_name: ‘Dave’, last_name: ‘Jungst’>,
        <#Person id: 3, first_name: ‘Dave’, last_name: ‘Davidson’>]>
```

Advanced Retrieving Records

- order

  => Person.all.order(:first_name)

  => Person.where(first_name: ‘Dave’).order(last_name: :desc)
- limit

  => Person.all.limit(3)
- group

  => Person.all.group(:first_name)
- not

  => Person.all.not(last_name: ‘Jungst’)

  => Person.all.where(first_name: ‘Dave’).not(last_name: ‘Jungst’)

Creating Records

```
Person.create(first_name: ‘Justin’, last_name: ‘Bieber’)
    => #<Person id: 4, first_name: ‘Justin’, last_name: ‘Bieber’>

p = Person.new(first_name: ‘Katy’, last_name: ‘Perry’)
    => #<Person id: nil, first_name: ‘Katy’, last_name: ‘Perry’>

p.save
    => #<Person id: 5, first_name: ‘Katy’, last_name: ‘Perry’>
```

Updating Records

- update(id,attributes):

  invoke model based validation, save the object when validation passes successfully else object is not save.

- update_attribute:

  This method update single attribute of object without invoking model based validation.

- update_attributes:

  This method update multiple attribute of single object and also pass model based validation.

Updating Records Examples

update(id,attributes):

```
Person.update(1, name: “Dave”, age: 32)
```

update_attribute:

```
dave = Person.find_by(id: params[:id])
dave.update_attribute(:race, “white”)
```

update_attributes:

```
attributes = jake = Person.find(params[:id])
obj.update_attributes({:name => “xyz”, :age => 20})
```

Deleting Records

Delete a single record

```
Person.find(1).destroy
```

Delete all records

```
Person.destroy_all
```

Delete all records that match

```
Person.where(first_name: ‘Dave’).destroy_all
```

Associations

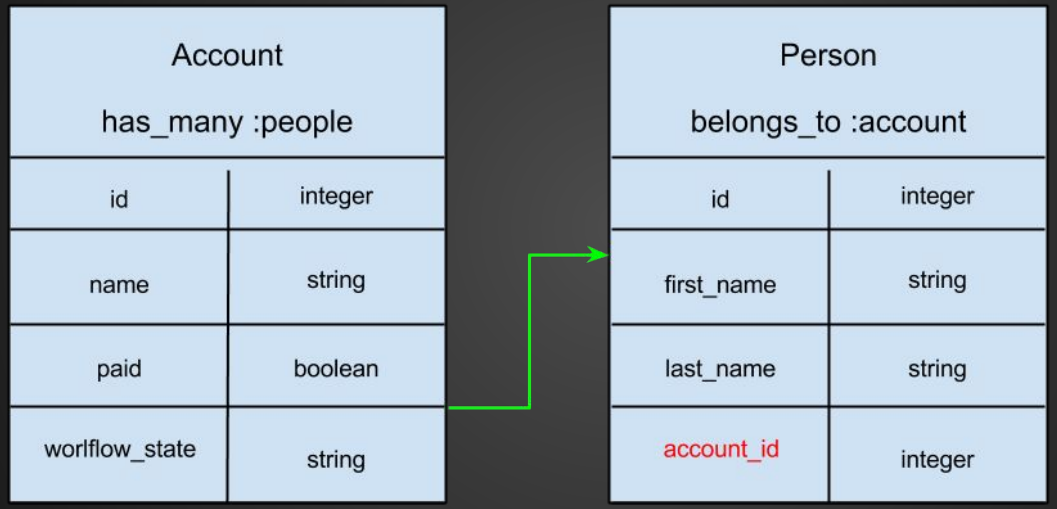
Types of associations

- belongs_to
- has_many
- has_one
- has_many :through
- has_one :through
- dependent

belongs_to

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

Person.create(first_name: ‘Steve’, last_name: ‘Jobs’, account_id: 1)
Account.find(1).person
      => #<Person id: 6 first_name: ‘Steve’, last_name: ‘Jobs’, account_id: 1>
Person.find(6).account
      => #<Account id:1 name: ‘Apple’>
Account.find(1).person.all
      => #<ActiveRecord::Relation [ …… ]>
```

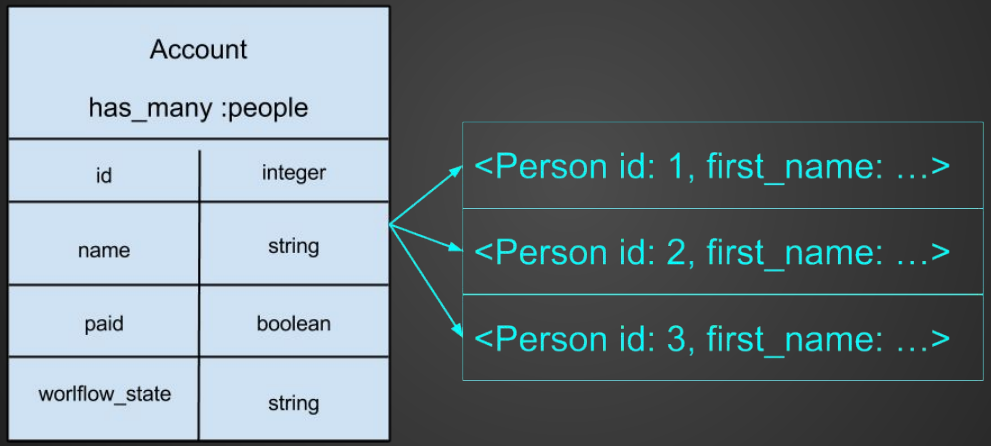
belongs_to


Account
has_many :people
id
integer
name
string
paid
boolean
worlflow_state
string
Person
belongs_to :account
id
integer
first_name
string
last_name
string
account_id
integer

has_many

```
class Account < ActiveRecord::Base
  has_many :people
end

Account.find(1).person.all
      => #<ActiveRecord::Relation [
                #<Person id: 1, first_name: ‘Jake’, last_name: ‘Sorce’,
                #<Person id: 2, first_name: ‘Dave’, last_name: ‘Jungst’
            ]>
Person.where(account_id: 1)
```

has_many


Account
has_many :people
id
integer
name
string
paid
boolean
worlflow_state
string
<Person id: 1, first_name: …>
<Person id: 2, first_name: …>
<Person id: 3, first_name: …>

has_one

```
class Account < ActiveRecord::Base
  has_many :people
  has_one :address
end

class Address < ActiveRecord::Base
  belongs_to :account
end
```

address

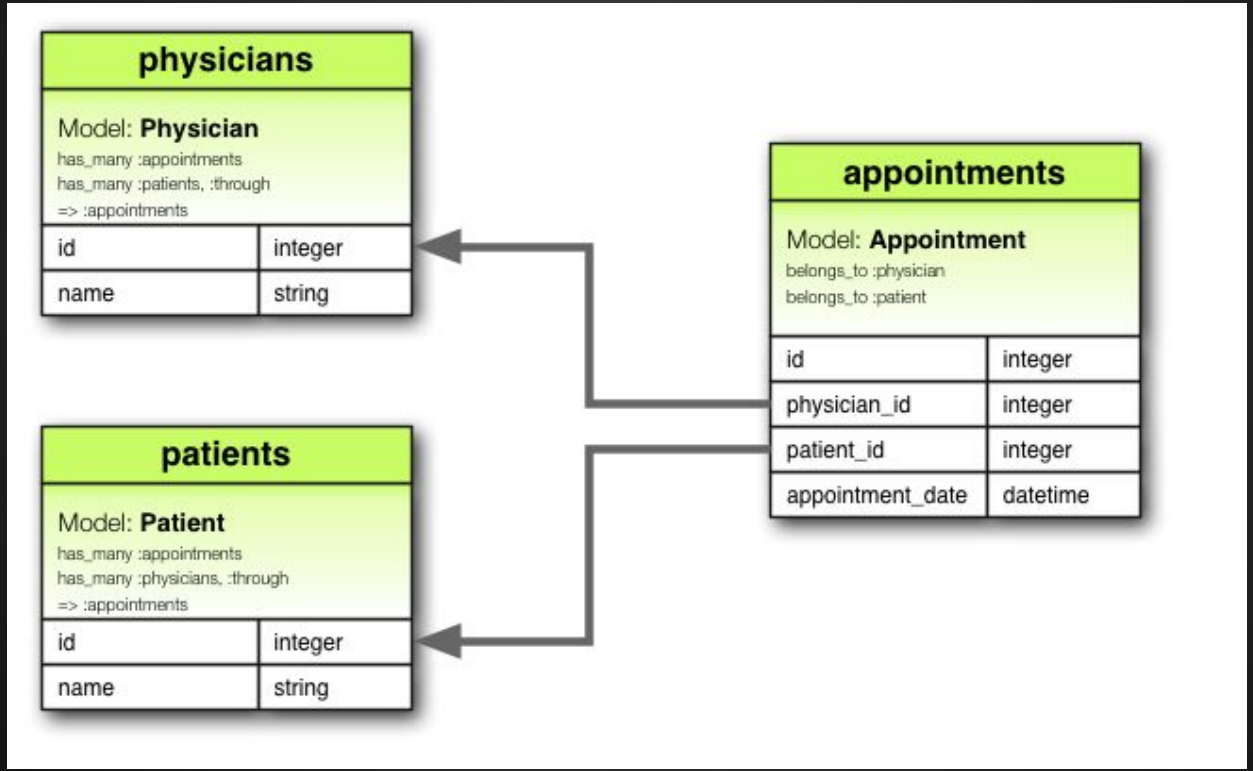
id	integer
city	string
state	string
zip	string
account_id	integer

has_many_through

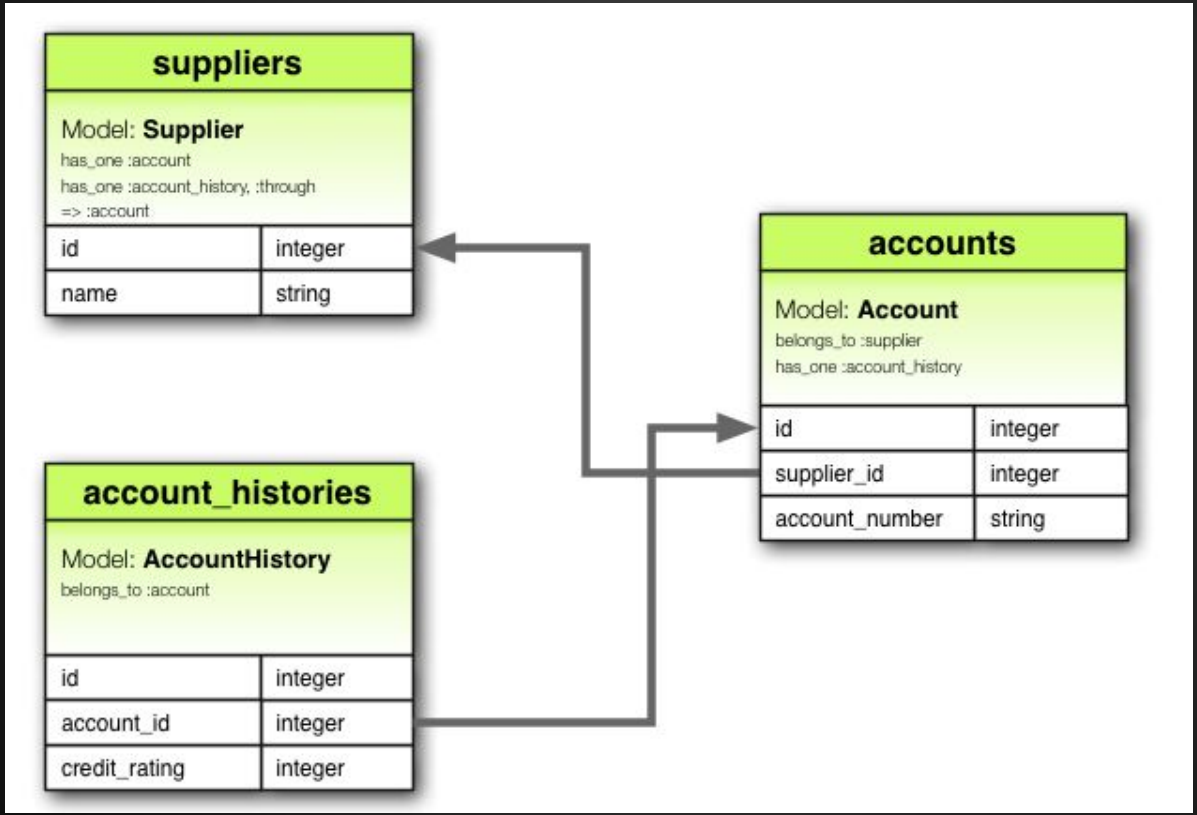
```ruby
class Physician < ActiveRecord::Base
  has_many :appointments
  has_many :patients, through: :appointments
end

class Patient < ActiveRecord::Base
  has_many :appointments
  has_many :physicians, through: :appointments
end
```

```ruby
class Appointment < ActiveRecord::Base
  belongs_to :physician
  belongs_to :patient
end
```

has_many_through



has_one_through



dependent

```
class Account < ActiveRecord::Base
  has_many :people, dependent: :destroy
end

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

When an account is destroyed it will automatically destroy all the people associated with it

```
Person.find(1)
    => #<Person id: 1, account_id: 1, …>

Account.find(1).destroy

Person.find(1)
    => nil
```

Validations

Most common validations:

- **confirmation**

  You should use this helper when you have two text fields that should receive exactly the same content.
- **inclusion**

  validates that the attributes' values are included in a given set.
- **length**

  validates the length of the attributes' values
- **numericality**

  validates that your attributes have only numeric values.
- **presence**

  validates that the specified attributes are not empty.
- **uniqueness**

  validates that the attribute's value is unique right before the object gets saved.

Validation Examples

Most common validations:

- **confirmation**

  Model:

```
class Person < ActiveRecord::Base
  validates :email, confirmation: true
end
```

  View:

```
<%= text_field :person, :email %>
<%= text_field :person, :email_confirmation %>
```

Validation Examples Cont...

Most common validations:

- **inclusion**

```
class Coffee < ActiveRecord::Base
  validates :size, inclusion: { in: %w(small medium large),
  message: "%{value} is not a valid size" }
end
```

- **length**

```
class Person < ActiveRecord::Base
  validates :name, length: { minimum: 2 }
  validates :bio, length: { maximum: 500 }
  validates :password, length: { in: 6..20 }
  validates :registration_number, length: { is: 6 }
end
```

Validation Examples Cont...

Most common validations:

- **numericality**

```
class Player < ActiveRecord::Base
  validates :points, numericality: true
  validates :games_played, numericality: { only_integer: true }
end
```

- **presence**

```
class Person < ActiveRecord::Base
  validates :name, :login, :email, presence: true
end
```

Validation Examples Cont...

Most common validations:

- **uniqueness**

```
class Account < ActiveRecord::Base
  validates :email, uniqueness: true
end
```

Validation Options

Most common validation options:

- **allow_nil**

  The :allow_nil option skips the validation when the value being validated is nil.

- **allow_blank**

  This option will let validation pass if the attribute's value is blank?, like nil or an empty string for example.

- **message**

  Lets you specify the message that will be added to the errors collection when validation fails.

- **on**

  The :on option lets you specify when the validation should happen.

Validation Options Examples

Most common validation options:

- **allow_nil**

```
class Coffee < ActiveRecord::Base
  validates :size, inclusion: { in: %w(small medium large),
    message: "%{value} is not a valid size" }, allow_nil: true
end
```

- **allow_blank**

```
class Topic < ActiveRecord::Base
  validates :title, length: { is: 5 }, allow_blank: true
end
```

Validation Options Examples Cont...

Most common validation options:

- **message**

```
class Coffee < ActiveRecord::Base
  validates :size, inclusion: { in: %w(small medium large),
  message: "%{value} is not a valid size" }
end
```

- **on**

```
class Person < ActiveRecord::Base
  validates :email, uniqueness: true, on: :create
  validates :age, numericality: true, on: :update
  validates :name, presence: true, on: :save
end
```

Callbacks

Callbacks allow you to trigger logic based on an event tied to a record

Some common callbacks are:

- before_validation
- after_validation
- before_save
- around_save
- before_create
- around_create
- after_create
- after_save

Callbacks in use

```
class CreditCard < ActiveRecord::Base
  before_save :encrypt_card_number

  private

  def encrypt_card_number
    self.card_number = bcrypt(self.card_number)
  end

end
```

When CreditCard.create(...) is called, before it is saved to the database it will first call the encrypt_card_number method

Attribute Serialization

The Migration:

```
class CreatePeople < ActiveRecord::Migration
  def change
    create_table :people do |t|
      t.belongs_to :account
      t.text :hobbies
    end
  end
end
```

The Model:

```
class Person < ActiveRecord::Base
  belongs_to :account
  serialize :hobbies
end
```

Model Methods

Class Methods

```
class Person < ActiveRecord::Base
  def self.by_first_name
    order(:first_name)
  end
end
```

Instance Methods

```
class Person < ActiveRecord::Base
  def full_name
    “#{self.first_name} {self.last_name}”
  end
end
```