Linguagem de Programação para Web

Ruby On Rails – parte 2 - Models

Prof. Tales Bitelo Viegas

Migrations

- Forma de alterar o Banco de Dados de uma maneira estruturada e organizada
- Mantém o registro de quais mudanças são necessárias na base de dados
- Gerenciado pelo framework ActiveRecord

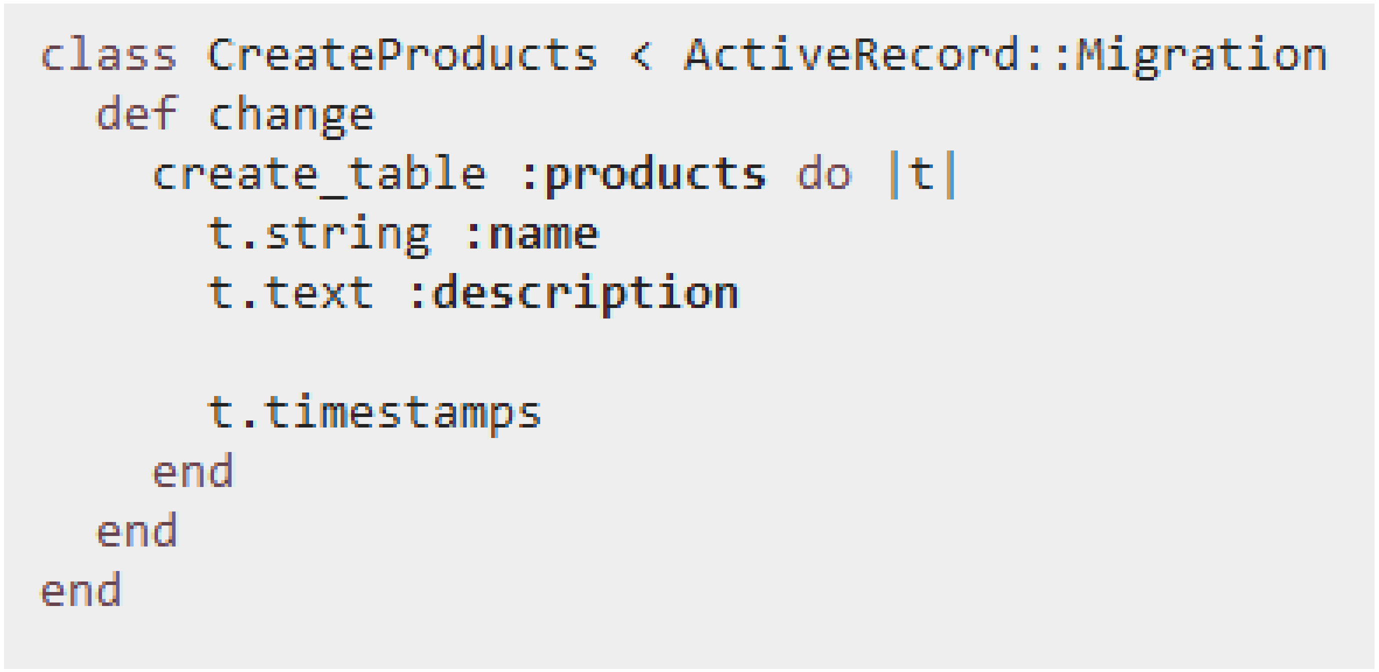
Migrations

```ruby
class CreateProducts < ActiveRecord::Migration
  def up
    create_table :products do |t|
      t.string :name
      t.text :description

      t.timestamps
    end
  end

  def down
    drop_table :products
  end
end
```

Migrations

```ruby
class AddReceiveNewsletterToUsers < ActiveRecord::Migration
  def up
    change_table :users do |t|
      t.boolean :receive_newsletter, :default => false
    end
    User.update_all ["receive_newsletter = ?", true]
  end

  def down
    remove_column :users, :receive_newsletter
  end
end
```

Migrations – Rails 3



```ruby
class CreateProducts < ActiveRecord::Migration
  def change
    create_table :products do |t|
      t.string :name
      t.text :description

      t.timestamps
    end
  end
end
```

Migrations

- Métodos para executar tarefas comuns:
  - add_column
  - add_index
  - change_column
  - change_table
  - create_table
  - remove_column
  - remove_index
  - rename_column
  - execute (executar um comando SQL)

Migrations

- Em bancos de dados que suportam transações para alteração de Schema, caso uma migration falhe, o rollback é executado

Migrations

- Tipos Suportados:
  - :binary
  - :boolean
  - :date
  - :datetime
  - :decimal
  - :float
  - :integer
  - :primary_key
  - :string
  - :text
  - :time
  - :timestamp

Criando um Modelo

Comando:

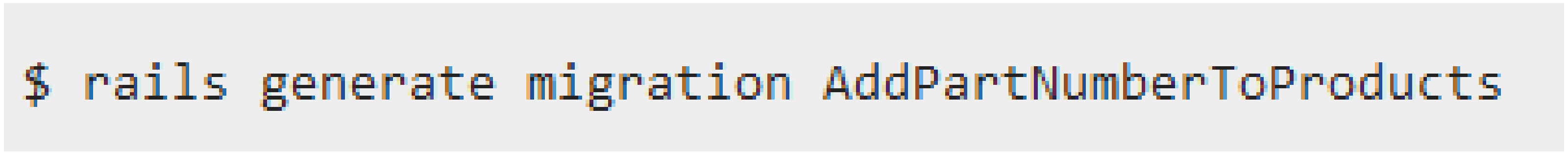
```
$ rails generate model Product name:string description:text
```

Gera a seguinte Migration:

```
class CreateProducts < ActiveRecord::Migration
  def change
    create_table :products do |t|
      t.string :name
      t.text :description

      t.timestamps
    end
  end
end
```

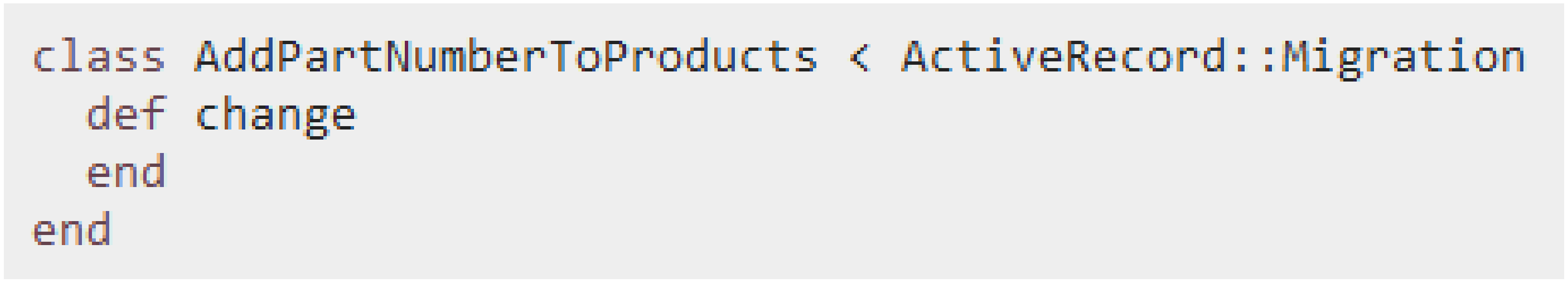
Criando uma Migration

- Comando:



```
$ rails generate migration AddPartNumberToProducts
```

- Gera a seguinte Migration vazia:



```
class AddPartNumberToProducts < ActiveRecord::Migration
  def change
  end
end
```

Criando uma Migration

- Comando

```
$ rails generate migration AddPartNumberToProducts part_number:string
```

- Gera a Migration já com o campo

```
class AddPartNumberToProducts < ActiveRecord::Migration
  def change
    add_column :products, :part_number, :string
  end
end
```

Rodando Migrations

- rake db:migrate
- rake db:migrate VERSION=20150226120000

- rake db:rollback
- rake db:rollback STEP=3

- rake rb:migrate:redo STEP=3

- rake db:reset

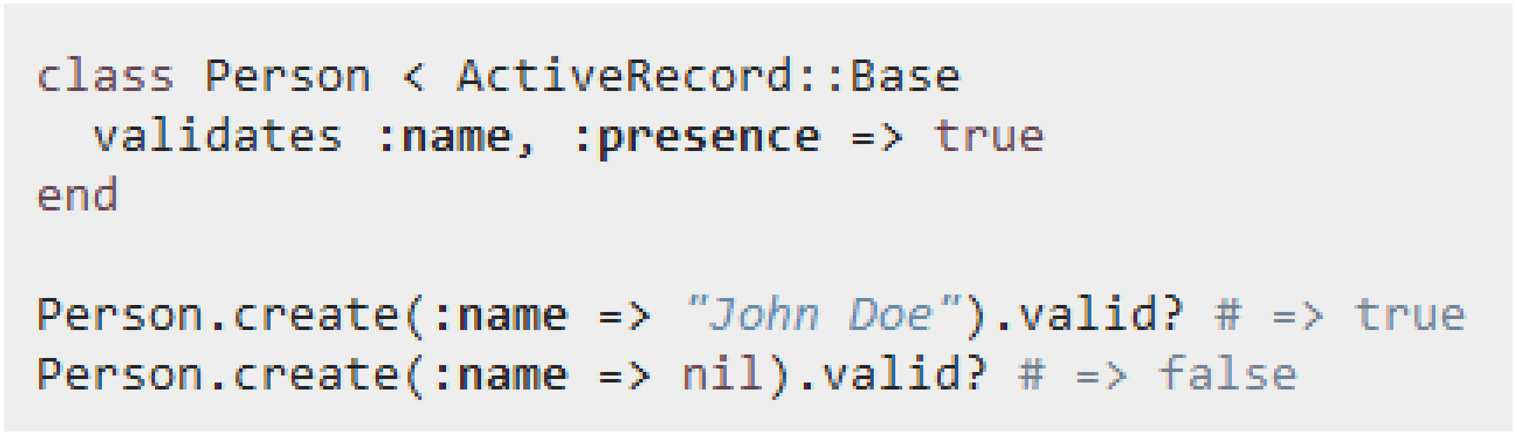
Validações

- Utilizadas para garantir que apenas dados válidos serão adicionados a base de dados
    - Database constraints
    - Client-side validations
    - Controller-level validations
    - Model-level validations

Validações

- valid? e invalid?

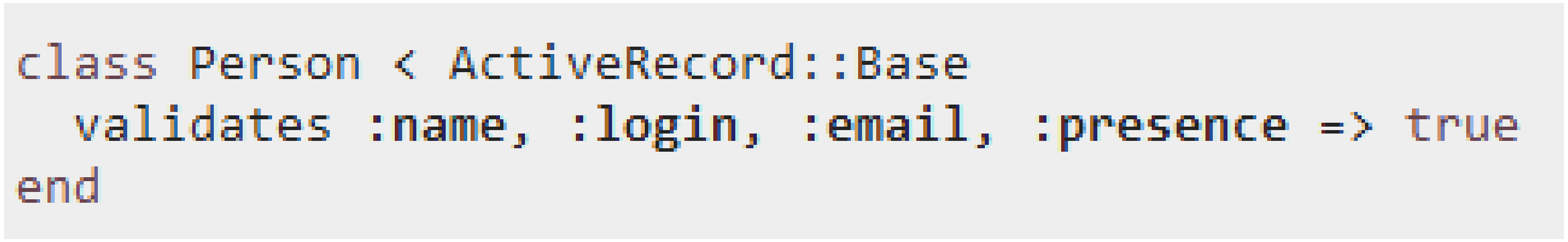
- rails generate model Person name:string login:string email:string password:string
- Usando o comando rails console



```ruby
class Person < ActiveRecord::Base
  validates :name, :presence => true
end

Person.create(:name => "John Doe").valid? # => true
Person.create(:name => nil).valid? # => false
```

Helpers

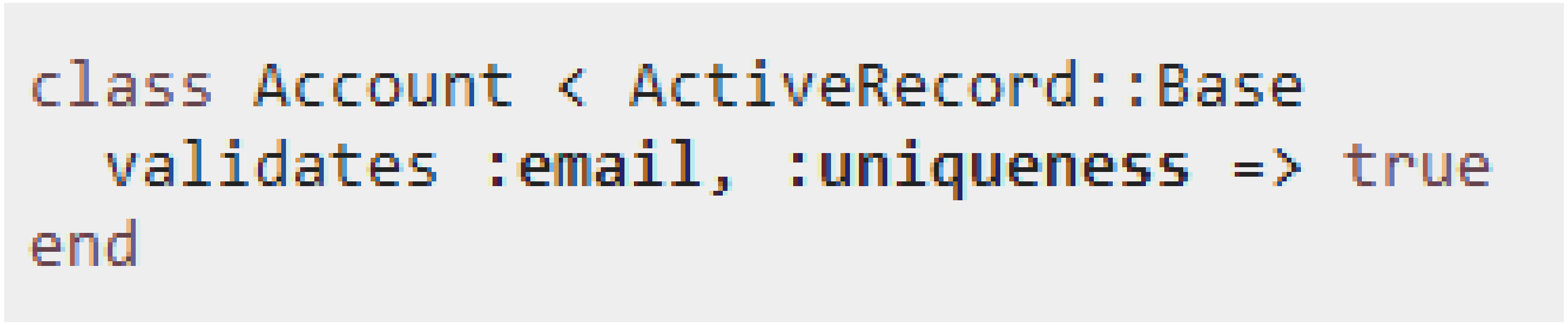
- presence
  - Verifica se o campo está presente



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

Helpers

- uniqueness
  - Valida a unicidade de um campo



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

Helpers

- acceptance
  - Valida se um checkbox na interface foi submetido.

```
class Person < ActiveRecord::Base
  validates :terms_of_service, :acceptance => true
end
```

Helpers

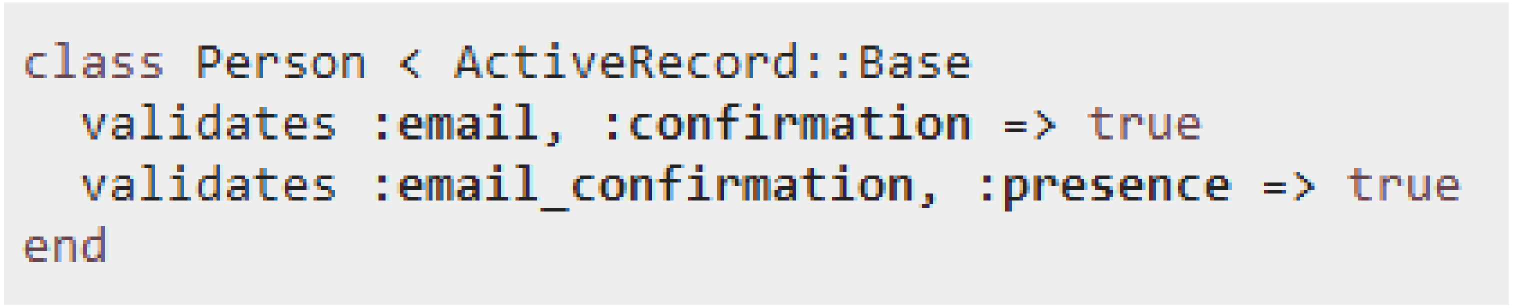
- confirmation
  - Dois campos que devem ter o mesmo valor

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

- Um campo deve ser email e o outro email_confirmation

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

Helpers



```
class Person < ActiveRecord::Base
  validates :email, :confirmation => true
  validates :email_confirmation, :presence => true
end
```

Helpers

- format
  - Valida se os valores estão em uma expressão regular

```
class Product < ActiveRecord::Base
  validates :legacy_code, :format => { :with => /\A[a-zA-Z]+\z/,
    :message => "Only letters allowed" }
end
```

Helpers

- length
  - Validam o tamanho
    - minimum
    - maximum
    - in
    - is

```ruby
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
```

Helpers

- numericality
  - Valida formatos numéricos

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

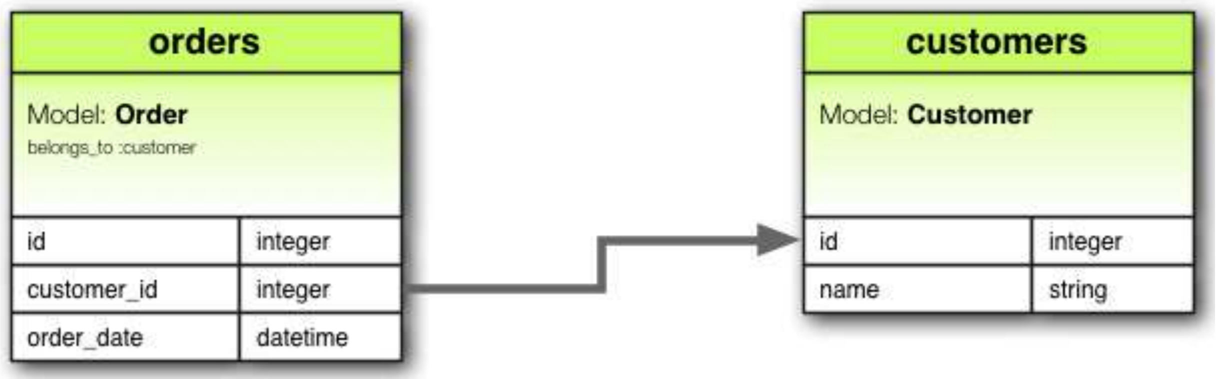
Associações

- Maneira como definimos os relacionamentos entre os modelos
  - belongs_to
  - has_one
  - has_many
  - has_many :through
  - has_one :through
  - has_and_belongs_to_many

Associações

- belongs_to
  - Define um relacionamento um para um entre o filho e o pai

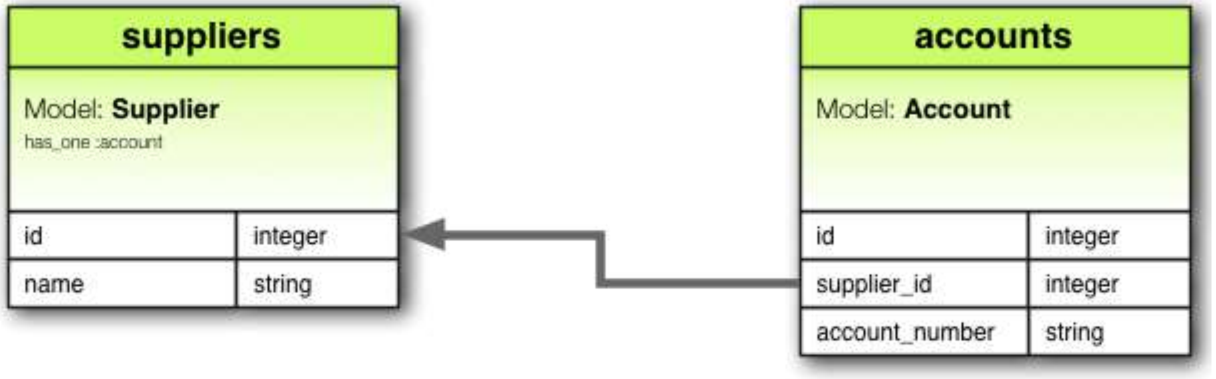
```
class Order < ActiveRecord::Base
  belongs_to :customer
end
```



Associações

- has_one
  - Define um relacionamento um para um entre o pai e o filho

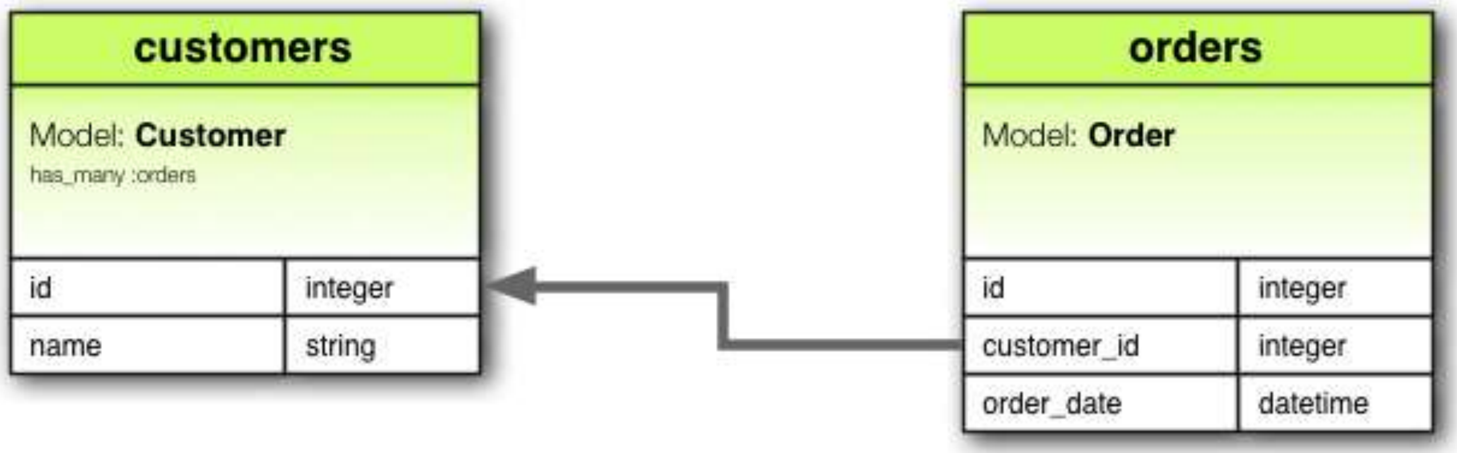
```
class Supplier < ActiveRecord::Base
  has_one :account
end
```



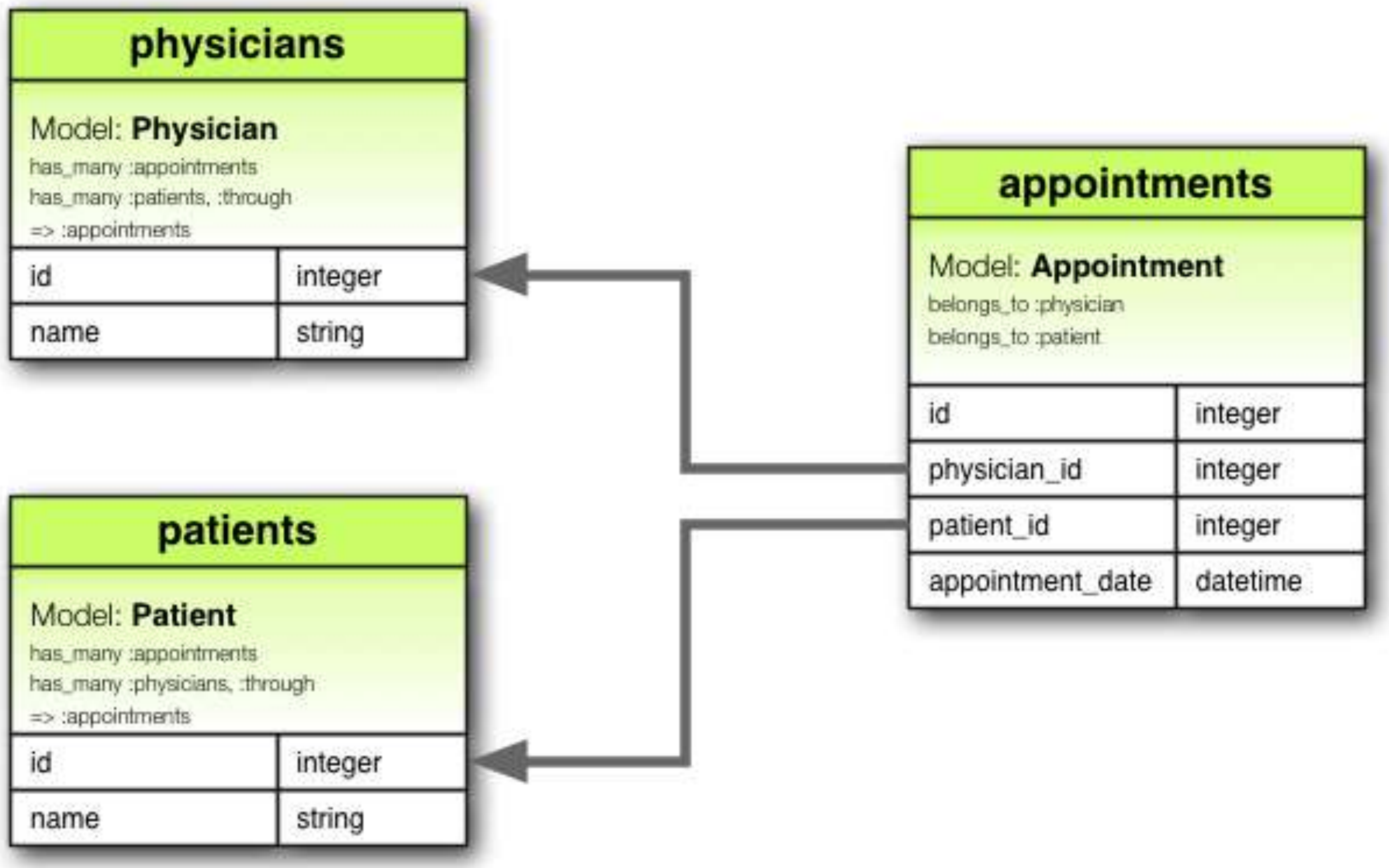
Associações

- has_many
  - Associação 1 para muitos

```
class Customer < ActiveRecord::Base
  has_many :orders
end
```

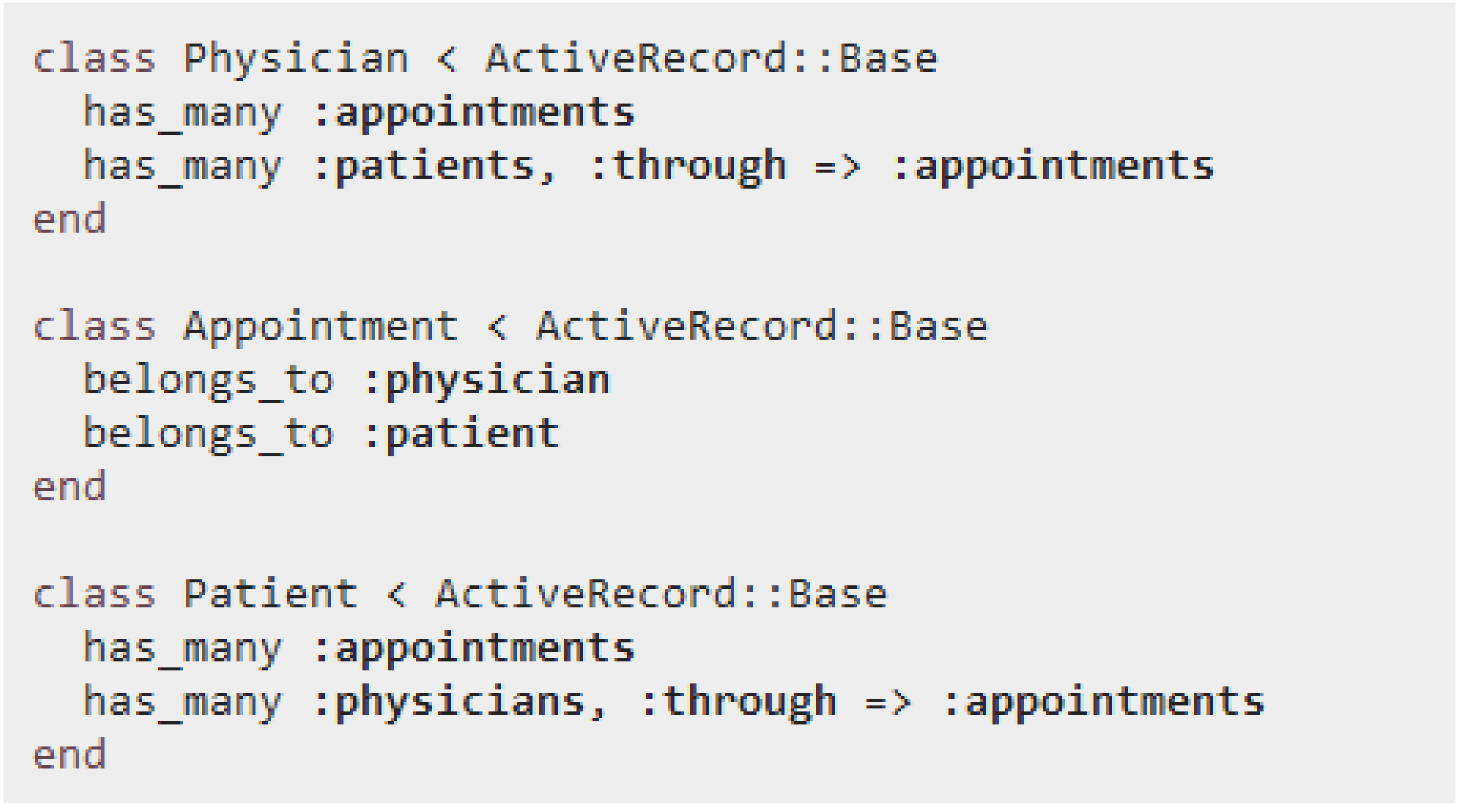


Associações



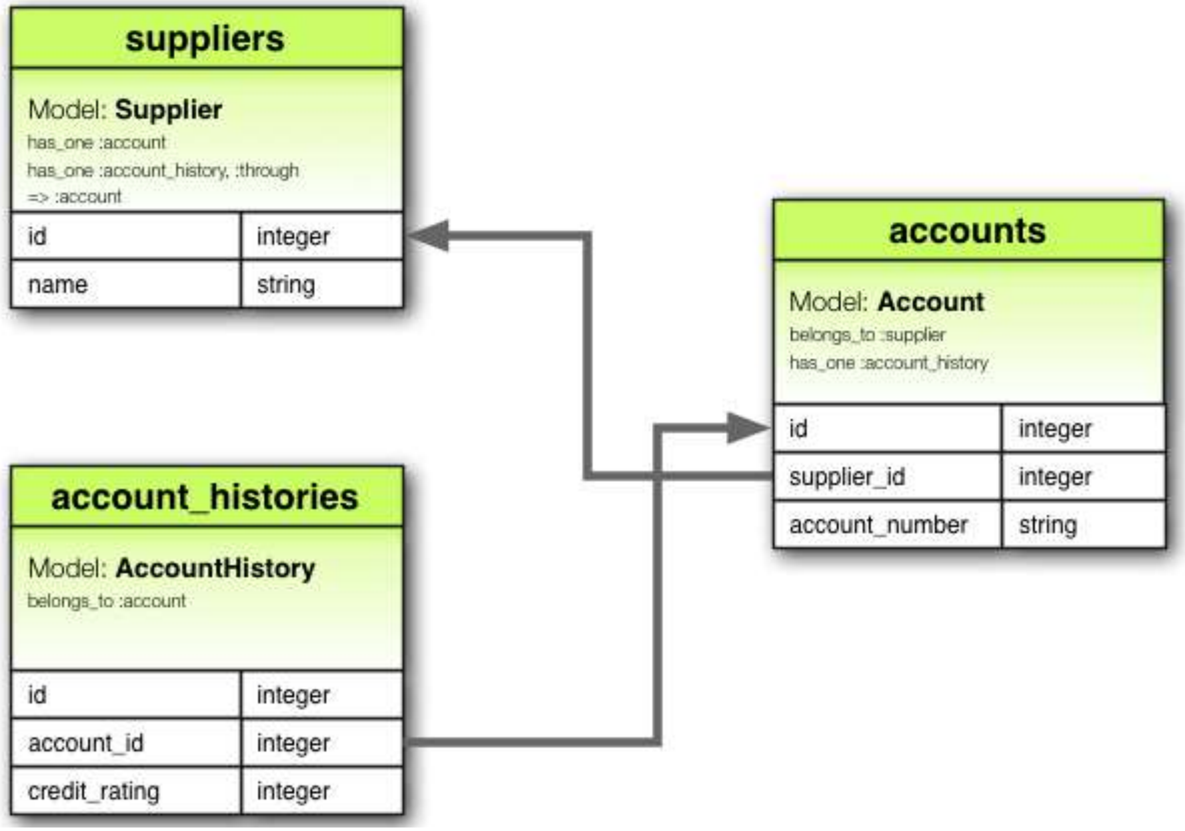
Associações

- has_many :through



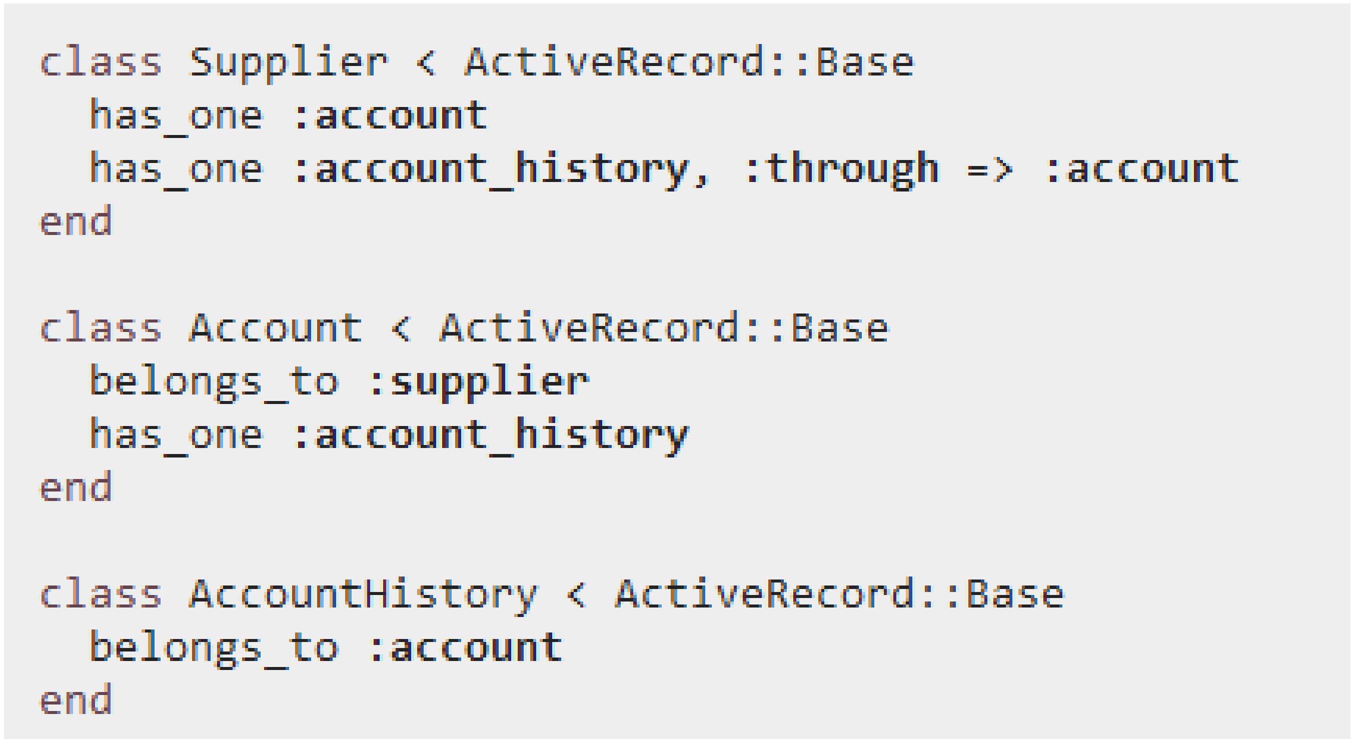
```ruby
class Physician < ActiveRecord::Base
  has_many :appointments
  has_many :patients, :through => :appointments
end

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

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

Associações



Associações

- has_one :through

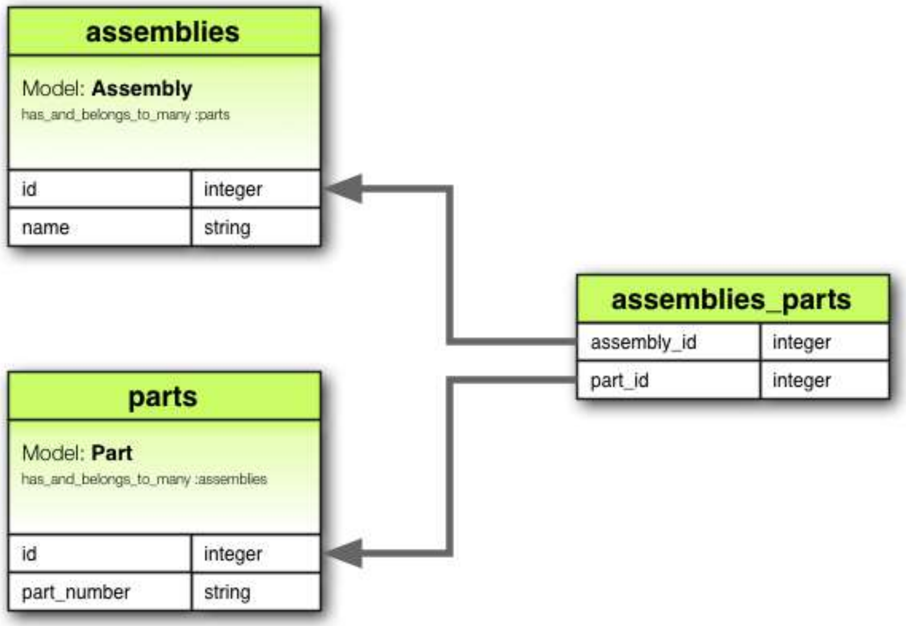


```ruby
class Supplier < ActiveRecord::Base
  has_one :account
  has_one :account_history, :through => :account
end

class Account < ActiveRecord::Base
  belongs_to :supplier
  has_one :account_history
end

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

Associações

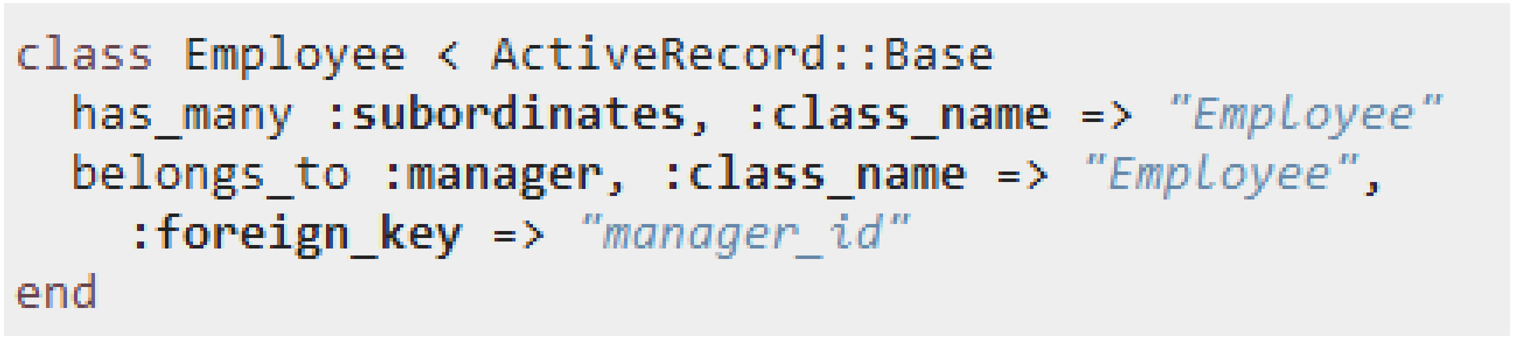
- has_and_belongs_to_many

```
class Assembly < ActiveRecord::Base
  has_and_belongs_to_many :parts
end

class Part < ActiveRecord::Base
  has_and_belongs_to_many :assemblies
end
```



Associações

- Auto-Relacionamentos



```
class Employee < ActiveRecord::Base
  has_many :subordinates, :class_name => "Employee"
  belongs_to :manager, :class_name => "Employee",
    :foreign_key => "manager_id"
end
```

Recomendações

- Crie as Foreign Keys para associações belongs_to

```
class Order < ActiveRecord::Base
  belongs_to :customer
end
```

```
class CreateOrders < ActiveRecord::Migration
  def change
    create_table :orders do |t|
      t.datetime :order_date
      t.string   :order_number
      t.integer  :customer_id
    end
  end
end
```

Recomendações

- Cria as tabelas de relacionamento para associações has_and_belongs_to_many

```
class Assembly < ActiveRecord::Base
  has_and_belongs_to_many :parts
end

class Part < ActiveRecord::Base
  has_and_belongs_to_many :assemblies
end
```

```
class CreateAssemblyPartJoinTable < ActiveRecord::Migration
  def change
    create_table :assemblies_parts, :id => false do |t|
      t.integer :assembly_id
      t.integer :part_id
    end
  end
end
```

Consultas com ActiveRecord

- Buscando um único Objeto
  - client = Client.find(10)
  - client = Client.first
  - client = Client.last

Consultas com ActiveRecord

- Buscando múltiplos objetos
  - client = Client.find([1, 10])
  - Client.all.each do |client|

```
                    puts client.id
         end
```

Condições

- Client.where(“orders_count = ?”, params[:orders])
- Client.where(“orders_count = ? AND locked = ?”, params[:orders], false)
- Client.where(“created_at >= :start_date AND created_at <= :end_date”, {:start_date => params[:start_date], :end_date => params[:end_date]})

Condições

- Client.where(:created_at => (params[:start_date].to_date)..(params[:end_date].to_date))

- Client.where(:locked => true)

- Client.where(:orders_count => [1.3.5])

Ordenação

- Client.order(“created_at”)
- Client.order(“created_at desc”)
- Client.order(“order_count asc, created_at desc)

Selecionando campos específicos

- Client.select(“viewable_by, locked”)
- Client.select(:name).uniq

Limite e Offset

- Client.limit(5)
- Client.limit(5).offset(30)

Join

- Category.joins(:posts)
- Post.joins(:category, :comments)
- Post.joins(:comments => :guest)

Find Dinâmicos

- find_by_<nome do campo>(valor)
- find_all_by_<nome do campo>(valor)
- find_last_by_<nome do campo>(valor)
- find_last_by_<nome do campo>_or_<nome do campo>(valor, valor)

Find by SQL

- Client.find_by_sql(“SELECT * FROM clients”)

Exists

- Client.exists?(1)
- Client.where(:first_name => ‘Ryan’).exists?
- Client.exists?

Cálculos

- Client.count
- Client.where(:fist_name => 'Ryan').count
- Client.average("orders_count")
- Client.minimum("age")
- Client.maximum("age")
- Client.sum("orders_count")