Reflecting on Active Record Associations

Daniel Colson

@composerinteralia
@dodecadaniel
class Repository < ApplicationRecord
  has_many :pull_requests
end

class PullRequest < ApplicationRecord
  belongs_to :repository
end
- repository
- repository=(repository)
- build_repository
- create_repository
- create_repository!
- reload_repository
- repository_changed?
- repository_previously_changed?
• repository
• repository=(repository)
• build_repository
• create_repository
• create_repository!
• reload_repository
• repository_changed?
• repository_previously_changed?

• Presence Validations
• Caching
• Autosaving
• Preloading
• Destroy Callbacks
• Scopes
• Extensions
• Etc.
Rails Magic
Let’s Study Magic!
Study Guide

• Metaprogramming
• Reflections
• Caching
• Relations
• Inverses
Metaprogramming
class PullRequest < ApplicationRecord
  belongs_to :repository
end
Metaprogramming

class PullRequest < ApplicationRecord
  self.belongs_to(:repository)
end

Calling a class method
Metaprogramming

class PullRequest < ApplicationRecord
  def self.belongs_to(name)
  end
end

Defining a class method
Metaprogramming

Reader Method
Metaprogramming

Reader Method

> pull_request
=> <#PullRequest @repository_id=42>
Metaprogramming

Reader Method

> pull_request
=> <#PullRequest @repository_id=42>

> pull_request.repository
=> <#Repository @id=42>
Metaprogramming

Reader Method

```ruby
> pull_request
=> <#PullRequest @repository_id=42>

> pull_request.repository
=> <#Repository @id=42>
```

Primary Key

Foreign Key

Primary Key
def self.belongs_to(name)
    def repository
        Repository.where(id: repository_id).first
    end
end
Metaprogramming

Writer Method
Metaprogramming

Writer Method

> pull_request
=> <#PullRequest @repository_id=42>
Metaprogramming

Writer Method

> pull_request
=> <#PullRequest @repository_id=42>

> repository
=> <#Repository @id=77>
Metaprogramming

Writer Method

> pull_request
=> <#PullRequest @repository_id=42>

> repository
=> <#Repository @id=77>

> pull_request.repository = repository
=> <#PullRequest @repository_id=77>
def self.belongs_to(name)
  def repository
    Repository.where(id: repository_id).first
  end

  def repository=(repository)
    self.repository_id = repository.id
  end
end
Metaprogramming

def self.belongs_to(name)
    def repository
        Repository.where(id: repository_id).first
    end

    def repository=(record)
        self.repository_id = repository.id
    end
end
Metaprogramming

def self.belongs_to(name)
    define_method(name) do
        Repository.where(id: repository_id).first
    end

    define_method("#{name}=") do |record|
        self.repository_id = repository.id
    end
end

Generic

Generic
def self.belongs_to(name)
    define_method(name) do
        Repository.where(:id => self[:repository_id]).first
    end
end

define_method("#{name}=") do |record|
    self[:repository_id] = record[:id]
end
end
def self.belongs_to(name)
  define_method(name) do
    klass.where(primary_key => self[foreign_key]).first
  end

  define_method("#{name}=") do |record|
    self[foreign_key] = record[primary_key]
  end
end
class PullRequest < ApplicationRecord
  belongs_to :repository
end
class PullRequest < ApplicationRecord
  belongs_to :repository
end

? klass       => Repository
  primary_key => :id
  foreign_key => :repository_id
Reflections

class Reflection
  def initialize(name, active_record)
    @active_record = active_record
    @name = name
  end
end
Reflections

Reflection

@active_record = PullRequest
@name = :repository
Reflections

Reflection

@active_record = PullRequest
@name = :repository

• #klass
• #primary_key
• #foreign_key
Reflections

Reflection

@active_record = PullRequest
@name = :repository

• #klass => Repository
• #primary_key
• #foreign_key
Reflections

- @active_record = PullRequest
- @name = :repository

- #klass => Repository
- #primary_key
- #foreign_key
class Reflection
  def klass
    @name.to_s.camelize.constantize
  end
end
Reflections

• #klass => Repository
• #primary_key => :id
• #foreign_key

Reflection

@active_record = PullRequest
@name = :repository
class Reflection
  def primary_key
    klass.primary_key
  end
end
Reflections

Reflection

@active_record = PullRequest
@name = :repository

• #klass => Repository
• #primary_key => :id
• #foreign_key => :repository_id
Reflections

Recall that a repository is a collection of pull requests. This example illustrates the relationship between a pull request and its associated repository.

```
@active_record = PullRequest
@name = :repository
```

- `#klass` => Repository
- `#primary_key` => :id
- `#foreign_key` => :repository_id
class Reflection
  def foreign_key
    "#{@name}_id"
  end
end
def self.belongs_to(name)
    reflection = Reflection.new(name, self)
def self.belongs_to(name)
    reflection = Reflection.new(name, self)
    klass = reflection.klass
    primary_key = reflection.primary_key
    foreign_key = reflection.foreign_key
def self.belongs_to(name)
  define_method(name) do
    klass.where(primary_key => self[foreign_key]).first
  end

  define_method("#{name}=") do |record|
    self[foreign_key] = record[primary_key]
  end
end
Caching 💰
Caching

> repo = pull_request.repository
=> #<Repository:0x307468 @id=42>

> same_repo = pull_request.repository
=> #<Repository:0x647570 @id=42>
Caching

```ruby
> repo = pull_request.repository
=> #<Repository:0x307468 @id=42>

> same_repo = pull_request.repository
=> #<Repository:0x647570 @id=42>
```

Different objects
Caching

> repo = pull_request.repository
=> #<Repository:0x307468 @id=42>

> same_repo = pull_request.repository
=> #<Repository:0x647570 @id=42>

Different objects

> repo.name = "new_name"
> same_repo.name
=> "old_name"
Caching

```ruby
> repo = pull_request.repository
=> #<Repository:0x307468 @id=42>

> same_repo = pull_request.repository
=> #<Repository:0x647570 @id=42>

> repo.name = "new_name"

> same_repo.name
=> "old_name"
```

Different objects

Inconsistent data
class Association
  def initialize(owner, reflection)
    @owner = owner
    @reflection = reflection
    @loaded = false
    @target = nil
  end
end
Caching

def self.belongs_to(name)
    define_method(name) do
        klass.where(#{primary_key} => self[foreign_key]).first
    end

    define_method("#{name}=") do |record|
        self[foreign_key] = record[primary_key]
    end
end
def self.belongs_to(name)
  define_method(name) do
  end

  define_method("#{name}=") do |record|
  end
end
Caching

class Association
  def reader
  end
  def writer(record)
  end
end

Association Reader

Association Writer
class Association

  def reader
    klass.where(primary_key => @owner[foreign_key]).first
  end

  def writer(record)
    @owner[foreign_key] = record[primary_key]
  end

end
Caching

```ruby
def self.belongs_to(name)
  define_method(name) do
    association(name).reader
  end

  define_method("#{name}=") do |record|
    association(name).writer(record)
  end
end
```

Use Association Reader and Writer
def self.belongs_to(name)
    define_method(name) do
        association(name).reader
    end

    define_method("#{name}=") do |record|
        association(name).writer(record)
    end
end

Use Association Reader and Writer

Get Association by name
Caching

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@owner</td>
<td>#&lt;PullRequest&gt;</td>
</tr>
<tr>
<td>@reflection</td>
<td>#&lt;Reflection&gt;</td>
</tr>
<tr>
<td>@loaded</td>
<td>false</td>
</tr>
<tr>
<td>@target</td>
<td>nil</td>
</tr>
</tbody>
</table>
Caching

def reader
  if loaded?
  else
  end
end

<table>
<thead>
<tr>
<th>Association</th>
</tr>
</thead>
<tbody>
<tr>
<td>@owner</td>
</tr>
<tr>
<td>@reflection</td>
</tr>
<tr>
<td>@loaded</td>
</tr>
<tr>
<td>@target</td>
</tr>
</tbody>
</table>
def reader
  if loaded?
    # association code
  else
    # association code
  end
end

Association

@owner = <PullRequest>
@reflection = <Reflection>
@loaded = false
@target = nil
Caching

def reader
    if loaded?
    else
        self.target = klass.where(...).first
    end
end
def reader
  if loaded?
    self.target = klass.where(...).first
  else
    self.target = klass.where(...).first
  end
end

Association

@owner = #=><PullRequest>
@reflection = #=><Reflection>
@loaded = true
@target = #=><Repository>
def reader
  if loaded?
    # code here
  else
    self.target = klass.where(...).first
  end
end

Association

@owner = #<PullRequest>
@reflection = #<Reflection>
@loaded = true
@target = #<Repository>
def reader
  if loaded?
    target
  else
    self.target = klass.where(...).first
  end
end
Caching

Beware of a stale cache!

def writer(record)
  @owner[foreign_key] = record[primary_key]
  self.target = record
end

Update target
Caching

> repo = pull_request.repository

=> #<Repository:0x776f6e @id=42>

> same_repo = pull_request.repository

=> #<Repository:0x776f6e @id=42>
Caching

> repo = pull_request.repository

=> #<Repository:0x776f6e @id=42>

> same_repo = pull_request.repository

=> #<Repository:0x776f6e @id=42>  

Same object
Caching

> repo = pull_request.repository
=> #<Repository:0x776f6e @id=42>

> same_repo = pull_request.repository
=> #<Repository:0x776f6e @id=42>

> repo.name = "new_name"
> same_repo.name
=> "new_name"
Caching

```ruby
> repo = pull_request.repository
=> #<Repository:0x776f6e @id=42>
> same_repo = pull_request.repository
=> #<Repository:0x776f6e @id=42>
> repo.name = "new_name"
> same_repo.name
=> "new_name"
```

Same object

Consistent data 👍
class Repository < ApplicationRecord
  has_many :pull_requests
end
def self.has_many(name)
  define_method(name) do
    association(name).reader
  end

  define_method("#{name}=") do |record|
    association(name).writer(record)
  end
end
Relations

class BelongsToAssociation < Association
end

class HasManyAssociation < Association
end
Relations

BelongsToAssociation#reader

def reader
  if loaded?
    target
  else
    self.target = klass.where(primary_key => @owner[foreign_key]).first
  end
end
Relations

BelongsToAssociation#reader

def reader
  if loaded?
    target
  else
    self.target = klass.where(primary_key => @owner[foreign_key]).first
  end
end
Relations

BelongsToAssociation#reader

def reader
  if loaded?
    target
  else
    self.target = klass.where(primary_key => @owner[foreign_key]).first
  end
end

Single Record

Primary Key

Foreign Key
Relations

HasManyAssociation #reader

def reader
  if loaded?
    target
  else
    self.target =
      klass.where(foreign_key => @owner[primary_key]).to_a
  end
end
Relations

HasManyAssociation #reader

def reader
    if loaded?
        target
    else
        self.target = klass.where(foreign_key => @owner[primary_key]).to_a
    end
end
def reader
  if loaded?
    target
  else
    self.target =
      klass.where(foreign_key => @owner[primary_key]).to_a
  end
end
Relations

> repository.pull_requests

=> SELECT * FROM pull_requests WHERE repository_id = 42

=> [#<PullRequest @repository_id=42>]

>
Relations

> repository.pull_requests
=> SELECT * FROM pull_requests WHERE repository_id = 42
=> [#<PullRequest @repository_id=42>]

> repository.pull_requests
=> [#<PullRequest @repository_id=42>]
Relations

What is a Relation?
Relations

What is a Relation?

> pull_requests = PullRequest.where(repository_id: 42)
=> [#ActiveRecord::Relation]
Relations

What is a Relation?

> pull_requests = PullRequest.where(repository_id: 42)
=> #ActiveRecord::Relation

• Super-powered array of records
• Lazy loading
• #create
• And much more
Relations

Association Relations?

```ruby
> pull_requests = repository.pull_requests
=> #<ActiveRecord::Relation>
```
Relations

Association Relations?

> pull_requests = repository.pull_requests
=> <ActiveRecord::Relation>

<table>
<thead>
<tr>
<th>Association</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>@loaded</td>
<td>false</td>
</tr>
<tr>
<td>@target</td>
<td>nil</td>
</tr>
</tbody>
</table>
Relations

Association Relations?

> pull_requests = repository.pull_requests
=> <#ActiveRecord::Relation>

> pull_requests.to_a

<table>
<thead>
<tr>
<th>Association</th>
</tr>
</thead>
<tbody>
<tr>
<td>@loaded     = false</td>
</tr>
<tr>
<td>@target     = nil</td>
</tr>
</tbody>
</table>
Relations

Association Relations?

```ruby
> pull_requests = repository.pull_requests
=> <ActiveRecord::Relation>

> pull_requests.to_a
=> [<PullRequest>]
```

Association

- @loaded = true
- @target = [<PullRequest>]
class CollectionProxy < ActiveRecord::Relation
  def initialize(association)
    @association = association
    super
  end
end
def reader
    CollectionProxy.new(self)
end

Relations

HasManyAssociation #reader

Updated Reader
def load_target
  if loaded?
    target
  else
    self.target =
      klass.where(foreign_key => @owner[primary_key]).to_a
  end
end
Repository pull_requests

```
CollectionProxy
@association =
```

```
HasManyAssociation
@loaded = false
@target = nil
```
repository.pull_requests
repository.pull_requests.to_a
repository.pull_requests
repository.pull_requests.to_a => []
Relations

```
repository.pull_requests
repository.pull_requests.to_a => []
repository.pull_requests.create
```
Relations

```
repository.pull_requests
repository.pull_requests.to_a => []
repository.pull_requests.create => #<PullRequest>
```
Relations

```ruby
repository.pull_requests
repository.pull_requests.to_a => []
repository.pull_requests.create => #<PullRequest>
repository.pull_requests.to_a => [#<PullRequest>]
```

```
CollectionProxy
  @association =

HasManyAssociation
  @loaded = true
  @target = [#<PullRequest>]
```
Inverses 😊
Inverses

class Repository < ApplicationRecord
  has_many :pull_requests
end

class PullRequest < ApplicationRecord
  belongs_to :repository
end
Inverses

> pull_requests = repository.pull_requests
=> SELECT * FROM pull_requests WHERE repository_id = 42
Inverses

> pull_requests = repository.pull_requests
=> SELECT * FROM pull_requests WHERE repository_id = 42
> pull_requests.map(&:repository)
Inverses

> pull_requests = repository.pull_requests

=> SELECT * FROM pull_requests WHERE repository_id = 42

> pull_requests.map(&:repository)

=> SELECT * FROM repositories WHERE id = 42 LIMIT 1

=> SELECT * FROM repositories WHERE id = 42 LIMIT 1

=> SELECT * FROM repositories WHERE id = 42 LIMIT 1

=> SELECT * FROM repositories WHERE id = 42 LIMIT 1

=> SELECT * FROM repositories WHERE id = 42 LIMIT 1

🤔
Inverses

<table>
<thead>
<tr>
<th>BelongsToAssociation</th>
</tr>
</thead>
<tbody>
<tr>
<td>@owner = <code>&lt;PullRequest&gt;</code></td>
</tr>
<tr>
<td>@loaded = <code>false</code></td>
</tr>
<tr>
<td>@target = <code>nil</code></td>
</tr>
</tbody>
</table>
Inverses

pull_request.repository

<table>
<thead>
<tr>
<th>BelongsToAssociation</th>
</tr>
</thead>
<tbody>
<tr>
<td>@owner</td>
</tr>
<tr>
<td>@loaded</td>
</tr>
<tr>
<td>@target</td>
</tr>
</tbody>
</table>
Inverses

pull_request.repository

BelongsToAssociation

@owner = #<PullRequest>
@loaded = false
@target = nil

repository.pull_requests

HasManyAssociation

@owner = #<Repository>
@loaded = true
@target = [#<PullRequest>, ...]
**Inverses**

- `pull_request.repository`:
  - `@owner` = `<PullRequest>`
  - `@loaded` = `true`
  - `@target` = `[<PullRequest>, ...]`

- `repository.pull_requests`:
  - `@owner` = `<Repository>`
  - `@loaded` = `true`
  - `@target` = `[<PullRequest>, ...]`
# Inverses

<table>
<thead>
<tr>
<th>HasManyAssociation</th>
</tr>
</thead>
<tbody>
<tr>
<td>@owner</td>
</tr>
<tr>
<td>@loaded</td>
</tr>
<tr>
<td>@target</td>
</tr>
</tbody>
</table>
Inverses

@target.each do |record|

**HasManyAssociation**

- @owner = #<Repository>
- @loaded = true
- @target = [#<PullRequest>, ...]
Inverses

```ruby
@target.each do |record|
  #:<PullRequest>
```

---

**HasManyAssociation**

- @owner = #:<Repository>
- @loaded = true
- @target = [#<PullRequest>, ...]
Inverses

@target.each do |record|
  association = record.association(:repository)
</PullRequest>

HasManyAssociation

@owner = #<Repository>
@loaded = true
@target = [#<PullRequest>, ...]
Inverses

```ruby
@target.each do |record|
  association = record.association(:repository)
  association.target = @owner
```

### HasManyAssociation

- @owner = `<Repository>`
- @loaded = true
- @target = `[<PullRequest>, ...]`
Inverses

```ruby
@target.each do |record|
  association = record.association(:repository)
  association.target = @owner
end
```

**HasManyAssociation**

- @owner = <%=Repository>
- @loaded = true
- @target = [ <%=PullRequest>, ... ]
@target.each do |record|
  association = record.association(:repository)
  association.target = @owner
end
Inverses

Reflection

@active_record = Repository
@name = :pull_requests
Inverses

Reflection

@active_record = Repository
@name = :pull_requests

#inverse_name => :repository
Inverses

Reflection

@active_record = Repository
@name = :pull_requests

#inverse_name => :repository
Inverses

class Reflection
  def inverse_name
    @active_record.name.underscore
  end
end
@target.each do |record|
  association = record.association(:repository)
  association.target = @owner
end
Inverses

@target.each do |record|
  association = record.association(@reflection.inverse_name)
  association.target = @owner
end
Inverses

> pull_requests = repository.pull_requests

=> SELECT * FROM pull_requests WHERE repository_id = 42
Inverses

> pull_requests = repository.pull_requests
=> SELECT * FROM pull_requests WHERE repository_id = 42
> pull_requests.map(&:repository)
Inverses

```ruby
> pull_requests = repository.pull_requests
=> SELECT * FROM pull_requests WHERE repository_id = 42

> pull_requests.map(&:repository)

👏
```
Further Studies

• Through Associations, Polymorphic Associations, Scoping, Etc.
• github.com/rails/rails
  • activerecord/lib/active_record/associations.rb
  • activerecord/lib/active_record/reflection.rb
  • activerecord/lib/active_record/associations/*
• danieljamescolson.com/blog
Practical Applications

• Use Rails features
• Write less custom code
• Write more efficient code
• Avoid inconsistent data
• Code with confidence
Reflecting on Active Record Associations

Daniel Colson

@composerinteralia
@dodecadaniel