Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

ActiveRecord

ORM de Rails

Raúl Montes T.

Modelos

Active Record provee el mapping entre clases y tablas

- `class Book < ActiveRecord::Base`
- asocia clase Book con tabla books
- atributos “coinciden” con las columnas

```
book = Book.new
book.title = "Rails for Dummies"
book.published_at = "2010-03-22"
book.save
```

Convenciones

- clases singulares, tablas plurales
- clases CamelCase, tablas snake_case

Tabla	Clase
events	Event
people	Person
order_items	OrderItem

- todas las tablas tienen id (entero correlativo) como clave primaria

Agregar Datos

- método de clase `new` seguido de `save`
  - a `new` también se le pueden entregar los atributos como hash
- método de clase `create`

```
book = Book.new
book.title = "Rails for Dummies"
book.published_at = "2010-03-22"
book.save
Book.create(title: "Rails for Dummies",
published_at: "2010-03-22")
```

Acceso a los Datos

- Mediante muchos métodos:
  - `find(:id)`
  - `all, first, last`
  - `find_by(:conditions_hash)`
  - `where(:conditions_hash)`
- Se pueden encadenar:
  - `Book.where(title: “Rails for Dummies”).first`

Consultas más Complejas

- `select(:attr1, …)` para seleccionar atributos
- `joins(:association)` para hacer un join basado en una asociación
- Especificando la tabla en una condición de where con joins, entregando una lista de opciones en lugar de igualdad directa:

```
where(tags: {name: name_list})
```

Consultas más Complejas

- where con condición SQL
- Ejemplo

```
Book.where("title = 'Rails for Dummies'")
Book.where("published_at > '23-01-2010'
AND title LIKE '%Rails%'")
Book.where("published_at < ?", Time.now)
Book.where("published_at < :date”, date:
Time.now)
```

Validación

- la validación se especifica en el modelo
- sólo información validada ingresa a la BD
- para cada atributo se pueden especificar muchas validaciones, entregando opciones a cada una

```
class Book < ActiveRecord::Base
  validates :title, presence: true, length: { minimum: 3 }
  validates :published_at, presence: true
end
```

- si save falla (retorna false) se usa método errors para revisar los errores

Otras Validaciones Built-in

- presence
- uniqueness
- length
- format
- confirmation (para passwords)
- numericality
- inclusion/exclusion
- acceptance (boolean)

Ejemplos

```
validates :email,
 uniqueness: true,
 length: { within: 5..50 },
 format: { with: /^[^@][\w.-]+@[\w.-]+[.][a-z]
 {2,4}$/i }
```

La estructura general es:

```
validates :field, validator1: [options_hash or
boolean], validator2: [options_hash or boolean]
```

Detalles en http://guides.rubyonrails.org/active_record_validations.html

Callbacks

- similar a los triggers de BD
- se llaman antes o después de una operación con la tabla asociada
- `before_create`, `after_create`
- `before_save`, `after_save`
- `before_destroy`, `after_destroy`

Observaciones

- cualquier `before` que retorne `false` detiene la operación
- puede ser usado para validaciones especiales
- `after` puede ser usado, por ejemplo, para notificar a alguien al agregarse un registro
- No hay que abusar de ellos. En general es mejor crear clases que realicen estas operaciones alrededor de crear/guardar un objeto.

Asociaciones

- Rara vez el modelo consiste en una sola clase/tabla
- Por lo general las clases del modelo están relacionadas
- Correspondencia con vinculación de las tablas asociadas (claves foráneas)

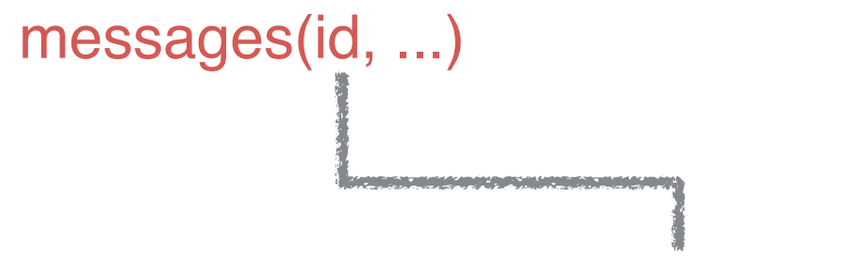
Nuevos Atributos de la Clase

- `has_one` - sirve para asociación 1:1
- `has_many` - 1:N
- `belongs_to` - N:1 o 1:1
- `has_and_belongs_to_many` - N:N

Ejemplos

```ruby
class Message < ActiveRecord::Base
  has_many :attachments
end

class Attachment < ActiveRecord::Base
  belongs_to :message
end
```




```ruby
class Category <ActiveRecord::Base
  has_and_belongs_to_many :articles
end

class Article <ActiveRecord::Base
  has_and_belongs_to_many :categories
end
```

Agregando Métodos al Modelo

- Múltiples usos
  - lógica de dominio
  - atributos calculados
  - cambio de formato/unidades de los datos
  - Por ejemplo, a Book podríamos agregarle

```
def long_title
  "#{title} – #{published_at}"
end
```

Manejo de transacciones

- Ejecutar las consultas en un bloque entregado al método transaction
- Cualquier excepción gatillará un Rollback y, salvo por ActiveRecord::Rollback, serán re-lanzadas

```
begin
  Resource.transaction do
    @resource.something_that_can_go_bad
    # usamos save! para que, si algo sale mal, lance
excepción
    @resource.save!
  end
rescue
  # capturamos la excepción
end
```