In this lecture, we will discuss…

- ✧ Active Record validations
- ✧ Writing custom validations

Validations

- Preferably, you would like to have some control over what goes into the database
- Not every input might be appropriate
- If these validations fail – your information should not be saved to the database
- Active Record provides a lot of built-in validators

:presence and :uniqueness

- `presence: true`
  - Make sure the field contains some data
- `uniqueness: true`
  - A check is performed to make sure no record exists in the database (already) with the given value for the specified attribute

:presence example

job.rb

```
1  class Job < ActiveRecord::Base
2    belongs_to :person
3    has_one :salary_range
4
5    validates :title, :company, presence: true
6
7  end
```

:presence example

```
~/advanced_ar$ rails c
Loading development environment (Rails 4.2.3)
irb(main):001:0> job = Job.new
=> #<Job id: nil, title: nil, company: nil, position_id: nil, person_id: nil, created_at: nil, updated_at: nil>
irb(main):002:0> job.errors
=> #<ActiveModel::Errors:0x007fe0b0b03590 @base=#<Job id: nil, title: nil, company: nil, position_id: nil, person_id:
 nil, created_at: nil, updated_at: nil>, @messages={}>
irb(main):003:0> job.save
   (0.2ms)  begin transaction
   (0.1ms)  rollback transaction
=> false
irb(main):004:0> job.errors
=> #<ActiveModel::Errors:0x007fe0b0b03590 @base=#<Job id: nil, title: nil, company: nil, position_id: nil, person_id:
 nil, created_at: nil, updated_at: nil>, @messages={:title=>["can't be blank"], :company=>["can't be blank"]}>
irb(main):005:0> job.errors.full_messages
=> ["Title can't be blank", "Company can't be blank"]
```

Other Common Validators

- `:numericality` – validates numeric input
- `:length` – validates value is a certain length
- `:format` – validates value complies with some regular expression format
- `:inclusion` – validates value is inside specified range
- `:exclusion` – validates value is out of the specified range

Writing Your Own Validator

1. Write a method that does some validation and calls `errors.add(columnname, error)` when it encounters an error condition
2. Specify it as a symbol for the `validate` method

Writing Your Own Validation

```
class SalaryRange < ActiveRecord::Base
  belongs_to :job

  validate :min_is_less_than_max

  def min_is_less_than_max
    if min_salary > max_salary
      errors.add(:min_salary, "cannot be greater than maximum salary!")
    end
  end
end
```

:numericality built-in validator can already do this for you – this is just to show an example of a custom validation…

Writing Your Own Validation

```
irb(main):001:0> sr = SalaryRange.create min_salary: 30000.00, max_salary: 10000.00
   (0.1ms)  begin transaction
   (0.0ms)  rollback transaction
=> #<SalaryRange id: nil, min_salary: 30000.0, max_salary: 10000.0, job_id: nil, created_at: nil, updated_at: nil>
irb(main):002:0> sr.errors
=> #<ActiveModel::Errors:0x007fe0acea87a8 @base=#<SalaryRange id: nil, min_salary: 30000.0, max_salary: 10000.0, job_
id: nil, created_at: nil, updated_at: nil>, @messages={:min_salary=>["cannot be greater than maximum salary!"]}>
irb(main):003:0> sr.errors.full_messages
=> ["Min salary cannot be greater than maximum salary!"]
irb(main):004:0> sr.save!
   (0.2ms)  begin transaction
   (0.0ms)  rollback transaction
ActiveRecord::RecordInvalid: Validation failed: Min salary cannot be greater than maximum salary!
```

Summary

- ✧ Validations give you control over what goes into DB
- ✧ See the guides for more information on **Active Record**
  - http://guides.rubyonrails.org/active_record_basics.html
  - http://guides.rubyonrails.orgactive_record_querying.html
  - http://guides.rubyonrails.org/association_basics.html
  - http://guides.rubyonrails.org/active_record_callbacks.html