In this lecture, we will discuss…

- ✧ One-to-Many Association
- ✧ Handling orphaned records

One-to-Many Association

- One person *has one or more* jobs
- One job entry *belongs to* exactly one person
- The "*belongs to*" side is the one with a foreign key

Convention: Default name for the foreign key is {master_table_singular}_id, e.g. person_id

Create Job Model and Migration

```
~/advanced_ar$ rails g model job title company position_id person:references
      invoke  active_record
      create    db/migrate/20150922141356_create_jobs.rb
      create    app/models/job.rb
      invoke    test_unit
      create      test/models/job_test.rb
      create      test/fixtures/jobs.yml
~/advanced_ar$ rake db:migrate
== 20150922141356 CreateJobs: migrating ==========================================
-- create_table(:jobs)
   -> 0.0020s
== 20150922141356 CreateJobs: migrated (0.0020s) =================================
```

Create Job Model and Migration

FOLDERS

- advanced_ar
  - app
  - bin
  - config
  - db
    - migrate
      - 20150908214851_create_people.r
      - 20150908221446_add_login_pass
      - 20150908232650_create_persona
      - 20150922141356_create_jobs.rb

20150922141356_create_jobs.rb

```ruby
class CreateJobs < ActiveRecord::Migration
  def change
    create_table :jobs do |t|
      t.string :title
      t.string :company
      t.string :position_id
      t.references :person, index: true, foreign_key: true

      t.timestamps null: false
    end
  end
end
```

Modifying Person and Job Models

FOLDERS

- advanced_ar
  - app
    - assets
    - controllers
    - helpers
    - mailers
    - models
      - concerns
      - .keep
      - job.rb
      - person.rb
      - personal_info.rb

person.rb

```ruby
class Person < ActiveRecord::Base
    has_one :personal_info
    has_many :jobs
end
```

job.rb

```ruby
class Job < ActiveRecord::Base
  belongs_to :person
end
```

Person and Job in Action

```
~/advanced_ar$ rails c
Loading development environment (Rails 4.2.3)
irb(main):001:0> ActiveRecord::Base.logger = nil
=> nil
irb(main):002:0> Job.create company: "MS", title: "Developer", position_id: "#1234"
=> #<Job id: 1, title: "Developer", company: "MS", position_id: "#1234", person_id: nil, created_at: "2015-09-22 14:30:4
>
irb(main):003:0> p1 = Person.first
=> #<Person id: 8, first_name: "Kalman", age: 33, last_name: "Smith", created_at: "2015-09-08 22:22:51", updated_at: "20
ss: "abc123">
irb(main):004:0> p1.jobs
=> #<ActiveRecord::Associations::CollectionProxy []>
irb(main):005:0> p1.jobs << Job.first
=> #<ActiveRecord::Associations::CollectionProxy [#<Job id: 1, title: "Developer", company: "MS", position_id: "#1234",
 14:30:49", updated_at: "2015-09-22 14:31:45">]>
irb(main):006:0> Job.first.person
=> #<Person id: 8, first_name: "Kalman", age: 33, last_name: "Smith", created_at: "2015-09-08 22:22:51", updated_at: "20
ss: "abc123">
irb(main):007:0>
```

More Methods

- `person.jobs = jobs`
  - Replaces existing jobs with a new array
  - As opposed to `person.jobs << job(s)` where the jobs are appended
- `person.jobs.clear`
  - Disassociates jobs from this person by setting the foreign key to NULL
- `create` and `where` methods for jobs become scoped to the `person`!

Scoped Jobs

FOLDERS
- advanced_ar
  - app
  - bin
  - config
  - db
    - migrate
    - development.sqlite3
    - schema.rb
    - seeds.rb
  - lib
  - log
  - public

seeds.rb

```ruby
Person.destroy_all

Person.create! [
]

Person.first.jobs.create! [
  { title: "Developer", company: "MS", position_id: "#1234" },
  { title: "Developer", company: "MS", position_id: "#1235" }
]

Person.last.jobs.create! [
  { title: "Sr. Developer", company: "MS", position_id: "#5234" },
  { title: "Sr. Developer", company: "MS", position_id: "#5235" }
]
```

There is also a build version, which does not automatically save to DB

```
~/advanced_ar$ rake db:seed
~/advanced_ar$
```

Scoped Jobs: Where

```
~/advanced_ar$ rails c
Loading development environment (Rails 4.2.3)
irb(main):001:0> ActiveRecord::Base.logger = nil
=> nil
irb(main):002:0> Person.first.jobs.where(company: "MS").count
=> 2
irb(main):003:0> Person.last.jobs.where(company: "MS").count
=> 2
irb(main):004:0> Person.last.jobs.where(company: "MS").to_a
=> [#<Job id: 4, title: "Sr. Developer", company: "MS", position_id: "#5234", person_id: 21, cre
:19">, #<Job id: 5, title: "Sr. Developer", company: "MS", position_id: "#5235", person_id: 21,
:39:19">]
irb(main):005:0>
```

Options for has_many - :class_name

FOLDERS
- advanced_ar
  - app
    - assets
    - controllers
    - helpers
    - mailers
    - models
      - concerns
      - .keep
      - job.rb
      - person.rb

person.rb

```ruby
class Person < ActiveRecord::Base
    has_one :personal_info
    has_many :jobs
    has_many :my_jobs, class_name: "Job"
end
```

class_name: 'Modelname'

```
~/advanced_ar$ rails c
Loading development environment (Rails 4.2.3)
irb(main):001:0> Person.first.my_jobs
  Person Load (0.1ms)  SELECT  "people".* FROM "people"  ORDER BY "people"."id" ASC LIMIT 1
  Job Load (0.1ms)  SELECT "jobs".* FROM "jobs" WHERE "jobs"."person_id" = ?  [["person_id", 15]]
=> #<ActiveRecord::Associations::CollectionProxy [#<Job id: 2, title: "Developer", company: "MS", position_id: "#1234", person_id: 15,
2 14:39:19", updated_at: "2015-09-22 14:39:19">, #<Job id: 3, title: "Developer", company: "MS", position_id: "#1235", person_id: 15,
 14:39:19", updated_at: "2015-09-22 14:39:19">]>
```

:dependent

- `has_many`, `has_one` and `belongs_to` support `:dependent` option which lets you specify the fate of the association when the parent gets destroyed

1. `:delete` – remove associated object(s)
2. `:destroy` – same as above, but remove the association by calling `destroy` on it
3. `:nullify` – set the FK to NULL (leave the associated entity alone – just disassociate)

:dependent - Example

FOLDERS

- advanced_ar
  - app
    - assets
    - controllers
    - helpers
    - mailers
    - models
      - concerns
      - .keep
      - job.rb
      - person.rb

person.rb

```ruby
class Person < ActiveRecord::Base
    has_one :personal_info, dependent: :destroy
    has_many :jobs
    has_many :my_jobs, class_name: "Job"
end
```

:dependent in action

```
irb(main):001:0> mike = Person.find_by first_name: "Michael"
  Person Load (0.2ms)  SELECT  "people".* FROM "people" WHERE "people"."first_name" = ? LIMIT 1  [["first_name", "Michael"]]
=> #<Person id: 31, first_name: "Michael", age: 15, last_name: "Smith", created_at: "2015-09-22 15:06:15", updated_at: "2015-09-22 1
pass: "not_telling">
irb(main):002:0> mike.personal_info
  PersonalInfo Load (0.1ms)  SELECT  "personal_infos".* FROM "personal_infos" WHERE "personal_infos"."person_id" = ? LIMIT 1  [["per
=> #<PersonalInfo id: 13, height: 5.5, weight: 200.0, person_id: 31, created_at: "2015-09-22 15:06:15", updated_at: "2015-09-22 15:0
irb(main):003:0> mike.destroy
   (0.2ms)  begin transaction
  SQL (0.6ms)  DELETE FROM "personal_infos" WHERE "personal_infos"."id" = ?  [["id", 13]]
  SQL (0.1ms)  DELETE FROM "people" WHERE "people"."id" = ?  [["id", 31]]
   (1.4ms)  commit transaction
=> #<Person id: 31, first_name: "Michael", age: 15, last_name: "Smith", created_at: "2015-09-22 15:06:15", updated_at: "2015-09-22 1
pass: "not_telling">
irb(main):004:0> PersonalInfo.find 13
  PersonalInfo Load (0.2ms)  SELECT  "personal_infos".* FROM "personal_infos" WHERE "personal_infos"."id" = ? LIMIT 1  [["id", 13]]
ActiveRecord::RecordNotFound: Couldn't find PersonalInfo with 'id'=13
```

Summary

- ✧ One-to-Many uses `has_many` and `belongs_to`
- ✧ Handle “orphaned” associations by specifying `dependent` on the main association

What’s Next?

- ✧ Many-to-Many Association