- Knowing what you know already, if I asked you for the name of the studio that produced the movie with ID 3, you'd do this:

```
@movie = Movie.find_by(id: 3)
@studio = Studio.find_by(id: @movie.studio_id)
@studio.name
=> "Paramount"
```

- This works well enough, but as it turns out, there's an even better way! Rails supports a DSL (domain-specific language) for associating models. Wouldn't it be nice if we could do:

```
@movie = Movie.find_by(id: 3)
@movie.studio.name
=> "Paramount"
```

- We can! By keeping our domain model exactly the way it is, but add just a little bit of DSL code to our models:

```
class Movie < ActiveRecord::Base
  belongs_to :studio
end
```

```
class Studio < ActiveRecord::Base
  has_many :movies
end
```

- Many-to-many associations are common
- They can always be modeled as 2 one-to-many associations
- The trick is to identify the real-world "join model" in the middle
- You can use **`has_many :through`** to have Rails "connect the dots" for you

```
class Movie < ActiveRecord::Base

  has_many :roles
  has_many :actors, through: :roles

end
```

```
class Role < ActiveRecord::Base

  belongs_to :movie
  belongs_to :actor

end
```

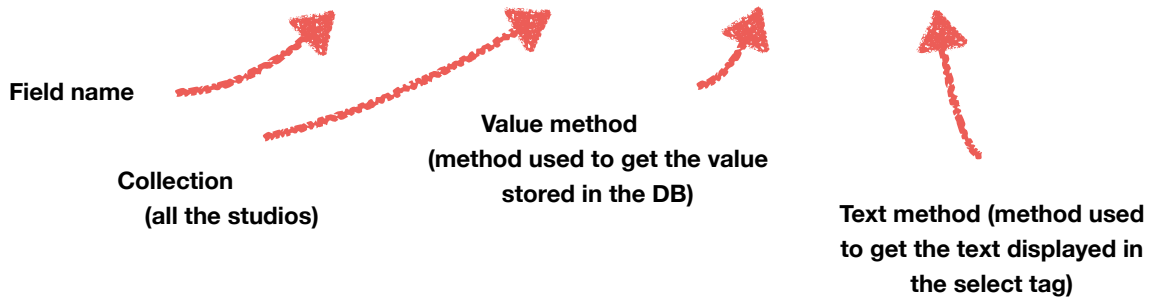

```
class Actor < ActiveRecord::Base

  has_many :roles
  has_many :movies, through: :roles

end
```

- Multi-model forms are common – for example, selecting the studio for a movie on its create or edit page.
- A common pattern is to use a drop-down, or in HTML-speak, a **select tag**.

```
<select>
  <option value="1">Paramount</option>
  <option value="2">Pixar</option>
  <option value="3">Fox</option>
  <option value="4">Lucasfilm</option>
</select>
```

**Select tag using Vanilla HTML**

```
<%= form.collection_select :studio_id, Studio.all, :id, :name %>
```




**Using the Rails collection_select form helper. Check http://api.rubyonrails.org/ for more options.**