In this lecture, we will discuss…

- Queries
- `find`
- `find_by`
- `find_or_create_by`
- `find_or_initialize_by`

`find`

- Find a document or multiple documents by their IDs
- Will raise an error by default if any of the IDs do not match
- `Actor.find("nm0993498").birth_name`
- `Actor.find("nm0000006", "nm0000008")`

find_by

- Find a document by the provided attributes
- `Movie.find_by(rated: "R")`
- `Movie.find_by(title: "Rocky")`

find_or_create_by

- Find a document by the provided attributes
- If not found, create and return a newly persisted one
- `Movie.find_or_create_by(title: "Titanic", year: "1997")`

find_or_initialize_by

- Find a document by the provided attributes
- If not found, initialize and return a new one
- `Movie.find_or_initialize_by(title: "Prisoners", year: "2013")`

Note: Does not persist

Summary

- `find` – like select in RDBMS and rich support in Mongoid

What's Next?

- `where`

In this lecture, we will discuss…

- `where`
- `count`
- `distinct`
- `exists`
- `geolocation`
- `first_or_create_by`
- `first_or_initialize_by`

where (count, distinct)

- `Movie.where(:title => "Rocky").count`
- `Movie.where(:year.gt => 2000).distinct(:title)`

Note: Schema Design lecture reference: year should be a number – we can use `'gt'` or `'lt'` queries

where (first)

- Get the first document
- If no sort options are provided, Mongoid will add an ascending _id sort to the criteria
- `Movie.first`
- `Movie.where(:rated=> "R").first`

where (exists?)

- Determine if any documents exist in the database
- Will return true for 1 or more
- `Movie.exists?`
- `Movie.where(:title => "Titanic").exists?`

where - :$exists and :$regex

- ```
  writer=Writer.where(:hometown=>{
          :$exists=>0}).first
  ```

- ```
  damon=Actor.where(:name=>{
        :$regex=>"Matt Da"}).first
  ```

`where` – Geolocation query

- In Mongoid, every model class has the built-in ability to express and execute a Geolocation query.

```
1  class Actor
2    include Mongoid::Document
3    include Mongoid::Timestamps
4
5    ..........
6
7    embeds_one :place_of_birth, class_name: 'Place' , as: :locatable
8
9    index ({ :"place_of_birth.geolocation" => Mongo::Index::GEO2DSPHERE })
10
11 end
```

`where` – Geolocation query

```
1  silver_spring=Place.where(:city=>"Silver Spring", :state=>"MD").first
2
3  Actor.near(:"place_of_birth.geolocation"=>silver_spring.geolocation)
4        .limit(5).each {|actor| pp "#{actor.name}, pob=#{actor.place_of_birth.id}"}
```

```
1  "Lewis Black, pob=Silver Spring, MD, USA"
2  "Jeffrey Wright, pob=Washington, DC, USA"
3  "Samuel L. Jackson, pob=Washington, DC, USA"
4  "Laura Cayouette, pob=Laurel, MD, USA"
5  "Mark Rolston, pob=Baltimore, MD, USA
```

where - first_or_create

- Find the first document by the provided attributes
- If not found, create and return a newly persisted one
- ```
  Movie.where(:title =>
  "Rocky20").first_or_create
  ```
  ```

where - first_or_initialize

- Find the first document by the provided attributes
- If not found, instantiate and return a new one
- ```
  Movie.where(:title =>
  "Rocky21").first_or_initialize
  ```
  ```

Summary

- ✧ `where` – with some additional criteria

What's Next?

- ✧ `pluck` and `scope`

In this lecture, we will discuss…

- `pluck`
- `scope`

`pluck`

- ✧ Get all the non nil values for the provided field
- ✧ `Movie.all.pluck(:title)`
- ✧ Entire document is not returned
  - Selected fields at the database level using a projection.

`Pluck` - Example

- `Movie.where(:rated=>"PG").map {|movie| [movie.title, movie.release_date]}`
  - Entire document is returned, grab 2 fields, rest is discarded
- `Movie.where(:rated=>"PG").pluck(:title, :release_date)`
  - Projection → 2 fields only, entire document is not returned
- `Movie.where(:title.lt=>"A").pluck(:title)`

scope

- Scopes provide a convenient way to reuse common criteria with more business domain style syntax
- Named scope
- Default scope

named scope

- Named scopes are simply criteria defined at class load that are referenced by a provided name

```
class Movie
    ...........
    field :year, type: Integer
    scope :current, ->{ where(:year.gt=>Date.current.year-2) }
end
```

- `Movie.current.where(:rated=>"R").pluck(:title, :year)`

default scope

- Default scopes can be useful when applying the same criteria to most queries, and you want something to be there by default
- Syntax :
  - `field :active, type: Boolean, default: true`

default scope

- Same criteria to most queries and something to be there by default

```ruby
class Airline
  include Mongoid::Document
  field :name, type: String
  field :active, type: Boolean, default: true

  default_scope ->{ where(active: true) }
end

Airline.each do |airline|
  # All airlines here are active.
end
```

default_scope

```ruby
airlineUA = Airline.create(
  name: "UNITED"
)

airlineLH = Airline.create(
  name: "LUFTHANSA"
)

airlinePA = Airline.create(
  name: "PANAM",
  active: false
)
```

- Airline.all.count = 2
  - *SELECT * from airlines where active = true*
- Airline.unscoped.all.count = 3
  - *SELECT * from airlines*

OR and in

- Union example with **in**:
  - `Movie.where(:year.gt => 2014).in(title: ["The Martian"]).pluck(:plot)`
- **or** conditional operator
  - `Movie.or({id: "tt3659388"},{title: "The Martian"}).pluck(:plot)`

Summary

- ✧ `scope` – convenient way to reuse common criteria
- ✧ `pluck` – get all the non nil values for the provided field
- ✧ `in` and `or`

What's Next?

- ✧ Mongoid – Scaffold

In this lecture, we will discuss…

- ✧ Scaffolding
- ✧ Movie Rails Application Demo

Basic Steps

- `> rails new movies`
- `> cd movies`
- `> gem 'mongoid', '~> 5.0.0'`
- `> bundle`
- `> rails g mongoid:config config/mongoid.yml`
- `> rails s`

Custom Classes and methods

- `Measurement`
- `Point`
- `initialize` - normalized form -- independent of source formats
- `to_s` - useful in producing formatted output
- `mongoize` – creates a DB form of the instance

Custom Classes – more methods

- `self.demongoize(object)` - creates an instance of the class from the DB-form of the data
- `self.mongoize(object)` - takes in all forms of the object and produces a DB-friendly form
- `self.evolve(object)` - used by criteria to convert object to DB-friendly form

SCAFFOLDING

- ✧ Note: mongoid is the default model generator.
  - To be explicit at command time, add the --orm mongoid option to the command line.

Place

- `Place` models a point and its descriptive address information.

```
$ rails g model Place formatted_address geolocation:Point street_number \
    street_name city postal_code county state country
```

Director

- `Director` models the detailed information of a movie director.
- `$ rails g model Director name`

DirectorRef

- `DirectorRef` is an annotated reference to a director that gets embedded into the `Movie`.
- `$ rails g model DirectorRef name`

Writer

- `Writer` holds the detailed information about the writer of a movie.
  - This class is directly associated with the movie without an annotated link.
- `$ rails g model Writer name`

Actor

- `Actor` contains the information details of an actor in a Movie.
- `$ rails g model Actor name birth_name date_of_birth:Date height:Measurement bio:text`

MovieRole

- `MovieRole` holds the role-specific information and relation between the Movie and Actor.
- `$ rails g model MovieRole character actor_name main:boolean url_character url_photo url_profile`

Movie

- `Movie` holds the core information about the movie, its properties, and supporting members.

```
$ rails g model Movie title type rated year:integer release_date:date \
    runtime:Measurement votes:integer countries:array languages:array \
    genres:array filming_locations:array metascore simple_plot:text \
    plot:text url_imdb url_poster directors:array actors:array
```

Controller/View

Controller and View - Assembly

```
$ rails g scaffold_controller Movie title type rated year:integer \
    release_date:date runtime:integer votes:integer countries:array \
    languages:array genres:array filming_locations:array metascore \
    simple_plot:text plot:text url_imdb url_poster directors:array actors:array
```

Summary

- Scaffolding and Assembly
- Demo

What's Next?

- Web Services