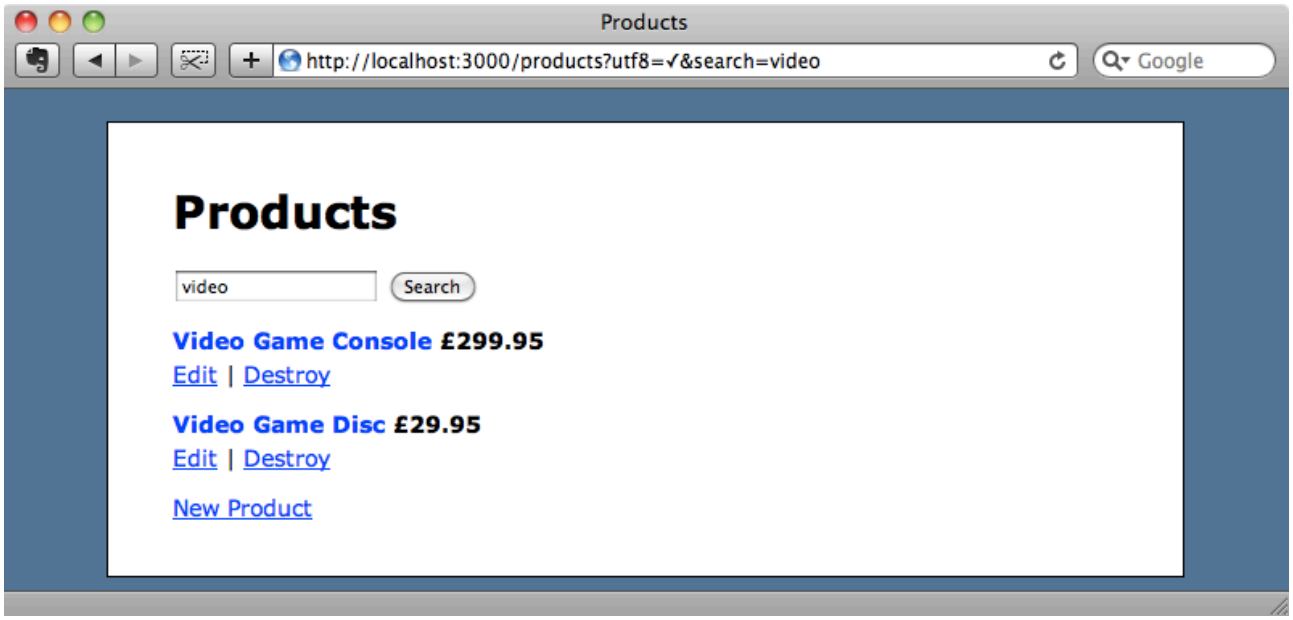
Episode 251

MetaWhere and MetaSearch

Below is a screenshot from a simple Rails 3 application that displays a list of products. The page has a search form that will filter the products by their name.



The search is performed in the `ProductsController`'s `index` action and the code looks like this:

/app/controllers/products_controller.rb

```
def index
  @products = Product.where("name LIKE ?", "%#{params[:search]}%")
end
```

The code uses the `where` method with a SQL snippet to find all of the products whose name is `LIKE` the search term. As the search condition is more complex than a simple equals we can't pass in a conditions hash and the same would apply if we were doing a less or greater than comparison. ActiveRecord requires SQL strings for all of these.

If you don't like having to use SQL fragments in your code there is a gem called MetaWhere[1] that you might find useful. MetaWhere allows you to call methods on symbols in the arguments passed to `where` so that you can make queries like this:

[1] http://metautonomo.us/projects/metawhere/

```
Article.where(:title.matches => 'Hello%', ↵
  :created_at.gt => 3.days.ago)
```

MetaWhere will generate the following SQL from this code:

```
SELECT "articles".* FROM "articles"
WHERE ("articles"."title" LIKE 'Hello%')
AND ("articles"."created_at" > '2010-04-12 18:39:32.592087')
```

With MetaWhere we can remove the need to use SQL fragments in queries and use hash conditions instead. This approach is similar to how DataMapper and Mongoid work and it's nice to have it as an option in ActiveRecord. To see how easy MetaWhere is to use we're going to modify our products application to use it.

First we'll need to add a reference to the gem in the application's `Gemfile`.

/Gemfile

```
# Edit this Gemfile to bundle your application's dependencies.
source 'http://gemcutter.org'

gem "rails", "3.0.3"
gem "sqlite3-ruby", :require => "sqlite3"
gem "meta_where"
```

We'll need to run the `bundle` command to make sure that the gem is installed. Once it is we can modify the search code in the `ProductsController` and remove the SQL fragment.

/app/controllers/products_controller.rb

```
def index
  @products = Product.where(:name.matches => ↵
    "%#{params[:search]}%")
end
```

With this simple change in place the application will behave as it did before and the same results will be returned for a given search.

Next we'll show some more complex examples in the console. For example we'll get all of the products whose price is less than five pounds.

```
ruby-1.9.2-p0 > Product.where(:price.lt => 5)
+----+----------------------+--------+-------------+------------+
| id | name                 | price  | released_at | updated_at |
+----+----------------------+--------+-------------+------------+
| 6  | 1 Pint of Milk       | 0.49   | 2010-06-06  | 2011-01-31 |
| 7  | Porridge Oats        | 1.99   | 2010-07-07  | 2011-01-31 |
+----+----------------------+--------+-------------+------------+
2 rows in set
```

We can also add OR conditions by using multiple hashes separated by the pipe symbol. The search below shows us all of the products that cost less than five pounds or which have the word “video” in their title.

```
ruby-1.9.2-p0 > Product.where({:price.lt => 5} | ↵
  {:name.matches => "%video%"})
+----+----------------------+--------+-------------+------------+
| id | name                 | price  | released_at | updated_at |
+----+----------------------+--------+-------------+------------+
| 6  | 1 Pint of Milk       | 0.49   | 2010-06-06  | 2011-01-31 |
| 7  | Porridge Oats        | 1.99   | 2010-07-07  | 2011-01-31 |
| 8  | Video Game Console   | 299.95 | 2010-08-08  | 2011-01-31 |
| 9  | Video Game Disc      | 29.95  | 2010-09-09  | 2011-01-31 |
+----+----------------------+--------+-------------+------------+
4 rows in set
```

MetaWhere also gives us some useful additions to the `order` method. If we want to order products by their `released_at` date to show us the most recently released products first we can do so with the following code.

```
ruby-1.9.2-p0 > Product.order(:released_at.desc)
+----+----------------------+--------+-------------+------------+
| id | name                 | price  | released_at | updated_at |
+----+----------------------+--------+-------------+------------+
| 9  | Video Game Disc      | 29.95  | 2010-09-09  | 2011-01-31 |
| 8  | Video Game Console   | 299.95 | 2010-08-08  | 2011-01-31 |
| 7  | Porridge Oats        | 1.99   | 2010-07-07  | 2011-01-31 |
| 6  | 1 Pint of Milk       | 0.49   | 2010-06-06  | 2011-01-31 |
| 5  | Oak Coffee Table     | 279.99 | 2010-05-05  | 2011-01-31 |
| 4  | Black Leather Sofa   | 499.99 | 2010-04-04  | 2011-01-31 |
| 3  | Stereolab T-Shirt    | 12.49  | 2010-03-03  | 2011-01-31 |
| 2  | DVD Player           | 79.99  | 2010-02-02  | 2011-01-31 |
| 1  | All-New Log For Girls | 29.95  | 2010-01-01  | 2011-01-31 |
+----+----------------------+--------+-------------+------------+
```

There's an alternative syntax we can use for `find` conditions but it needs to be enabled first. We can do this by calling

```
MetaWhere.operator_overload!
```

When this is enabled we can use standard Ruby operators instead of the `gt` and `lt` methods. To get all of the products costing less than £5 using this syntax we can run

```
ruby-1.9.2-p0 > Product.where(:price < 5)
+----+----------------------+--------+-------------+------------+
| id | name                 | price  | released_at | updated_at |
+----+----------------------+--------+-------------+------------+
| 6  | 1 Pint of Milk       | 0.49   | 2010-06-06  | 2011-01-31 |
| 7  | Porridge Oats        | 1.99   | 2010-07-07  | 2011-01-31 |
+----+----------------------+--------+-------------+------------+
2 rows in set
```

This gives us a more convenient way to define search conditions.

There is much more that can be done with the MetaWhere gem and there are it's well worth reading the documentation[2] to find out more.

MetaSearch

For the rest of this episode we'll take a look at another gem by the same author: MetaSearch[3]. This gem adds a convenient way to perform searches on a model through a form. With it we can call `search` on a model, and pass in search parameters from the form. We can then fetch the records that match that given query. An example of this is shown below.

```
def index
  @search = Article.search(params[:search])
  @articles = @search.all
end
```

In the view code the name of each field defines the search functionality. A text field

---

[2] http://metautonomo.us/projects/metawhere/

[3] http://metautonomo.us/projects/metasearch/

in the form called `title_contains`, for example, will mean that the search will search for records whose `title` contains the value entered into that text field.

Let's give this a try in our application by replacing our current search form with one that uses MetaSearch. The first step, as it was with MetaWhere, is to add a reference to the gem in the `Gemfile`.

/Gemfile

```
# Edit this Gemfile to bundle your application's dependencies.
source 'http://gemcutter.org'

gem "rails", "3.0.3"
gem "sqlite3-ruby", :require => "sqlite3"
gem "meta_where"
gem "meta_search"
```

We'll need to run `bundle` again to install the gem. Next we can replace the search code in the `ProductsController` with the equivalent MetaSearch code. That code currently looks like this:

/app/controllers/products_controller.rb

```
def index
  @products = Product.where(:name.matches => ↵
    "%#{params[:search]}%")
end
```

We'll replace it with this:

/app/controllers/products_controller.rb

```
def index
  @search = Product.search(params[:search])
  @products = @search.all
end
```

We create the search by calling `Product.search`, passing in the parameters from the form. Then we call `@search.all` to get a list of the matching products. This will fetch the products using an SQL query immediately, if we want to add additional conditions to the query afterwards we can user `relation` instead of `all` which will

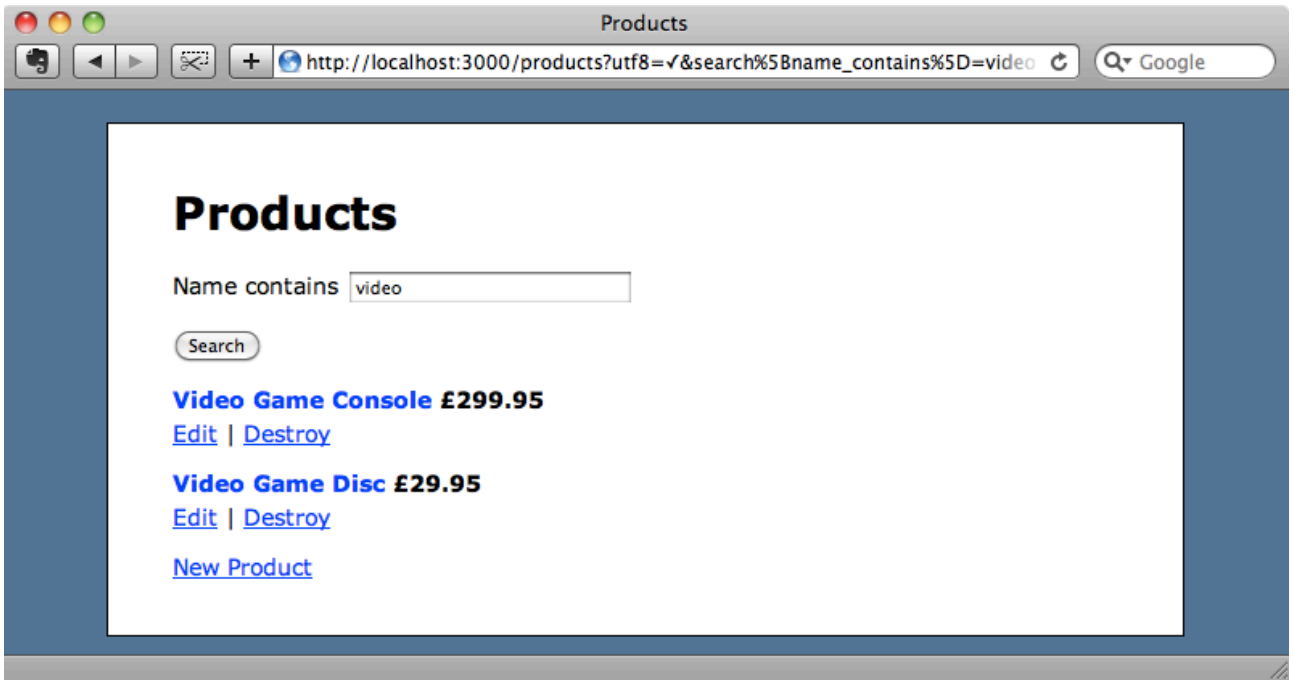
return a scope. If we needed pagination we could use `paginate` but in this case `all` will do what we need.

In the `index` template we need to update the search form to use the field names that MetaSearch requires. The form currently uses `form_tag`, but we'll need to use `form_for` instead with the search object. We want to search for products whose `name` contains the search term and so the text field on the form needs to be called `name_contains`.

/app/views/products/index.html

```
<%= form_for @search do |f| %>
  <p>
    <%= f.label :name_contains %>
    <%= f.text_field :name_contains %>
  </p>
  <p class="button"><%= f.submit "Search" %></p>
<% end %>
```

If we reload the products page now we'll see the new form and if we search for the same term we did before we'll get the same results.



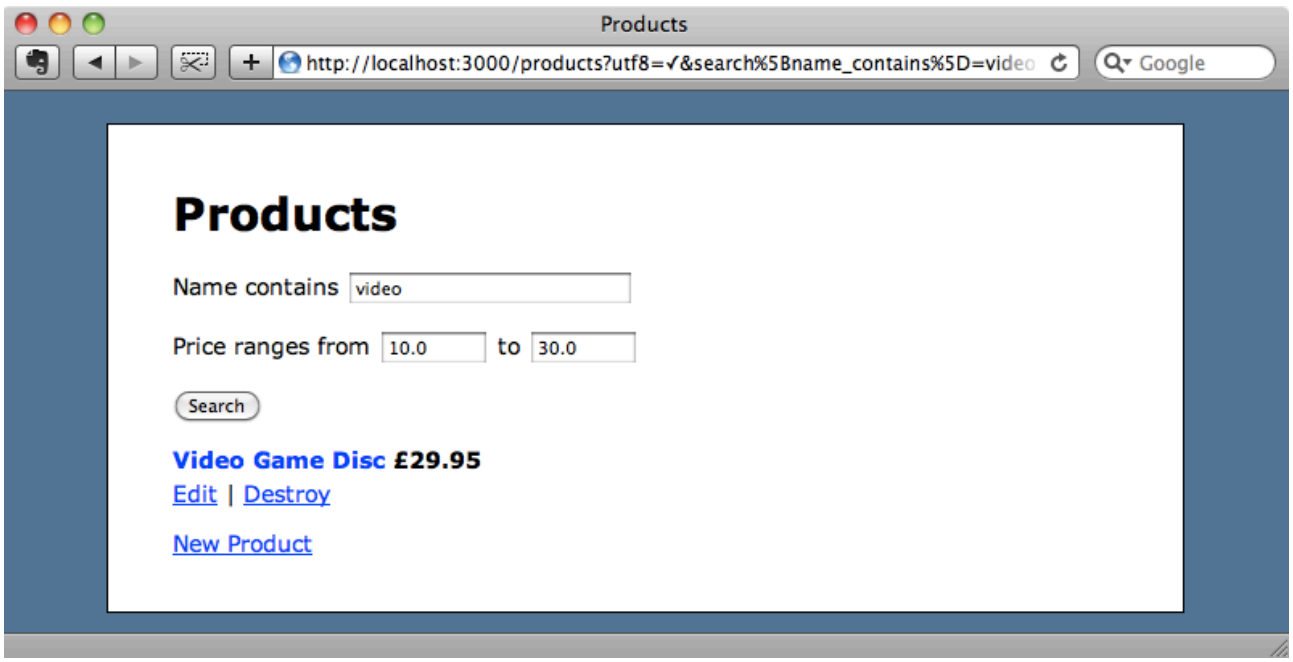
What's useful about this is that we can add expand the search options by simply adding more fields in the view layer, we don't need to modify the model or

controller layers at all. If, for example, we want to add the ability to search by a price range we just need to change the form and add text fields called `price_gte` and `price_lte` as follows:

/app/views/products/index.html.erb

```
<%= form_for @search do |f| %>
  <p>
    <%= f.label :name_contains %>
    <%= f.text_field :name_contains %>
  </p>
  <p>
    <%= f.label :price_gte, "Price ranges from" %>
    <%= f.text_field :price_gte, :size => 8 %>
    <%= f.label :price_lte, "to" %>
    <%= f.text_field :price_lte, :size => 8 %>
  </p>
  <p class="button"><%= f.submit "Search" %></p>
<% end %>
```

Reloading the page again will show the two new fields. We can now search for items by price range or by both name and price, so if we search for products with a name containing “video” that cost between ten and thirty pounds we’ll get the one matching product returned.



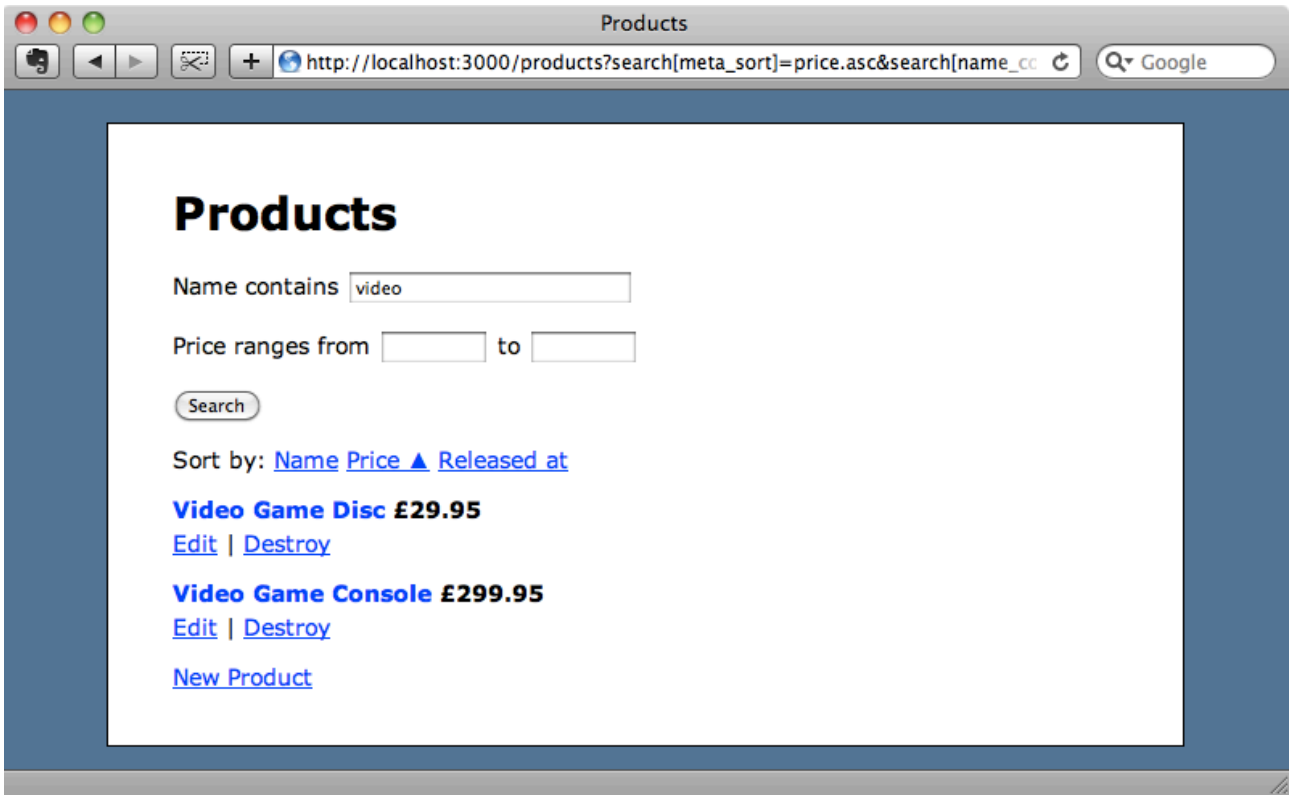
The key here is in the names given to the form fields as these determine the search conditions. There are more options that can be used in the form fields and these are covered by the documentation[4].

Another feature that MetaSearch provides is the ability to order the results by a given field. We can use a `sort_link` method that it provides. This method takes two arguments: a search object and a column name, so we can add sort fields with the following code:

/app/views/products/index.html.erb

```
<p>
  Sort by:
  <%= sort_link @search, :name %>
  <%= sort_link @search, :price %>
  <%= sort_link @search, :released_at %>
</p>
```

When we reload the page now we'll see the links and we can sort the products by clicking one of them. We can also combine the sorting with a filter.



[4] https://github.com/ernie/meta_search#readme

Security

One thing to be aware of when using MetaSearch is that it opens up our database tables to be searchable by any of their columns, simply by altering the form before it's submitted. This also applies to associations so if we have associated records that contain sensitive data then we'll need to be aware of this when using MetaSearch.

To help with this problem there are some security methods that we can add to our models to limit the fields that can be searched against. We won't go into details about this here but there are details about this on the MetaSearch website[5]. If you're using MetaSearch on a public-facing website then you'll definitely want to implement this.

That's it for our look at MetaWhere and MetaSearch. There is another gem that provides similar functionality called Searchlogic, but this doesn't work with Rails 3. If you like the behaviour of Searchlogic, but want a solution that works in Rails 3 then MetaWhere and MetaSearch are definitely worth a look.

---

[5] http://metautonomo.us/projects/metasearch/#security