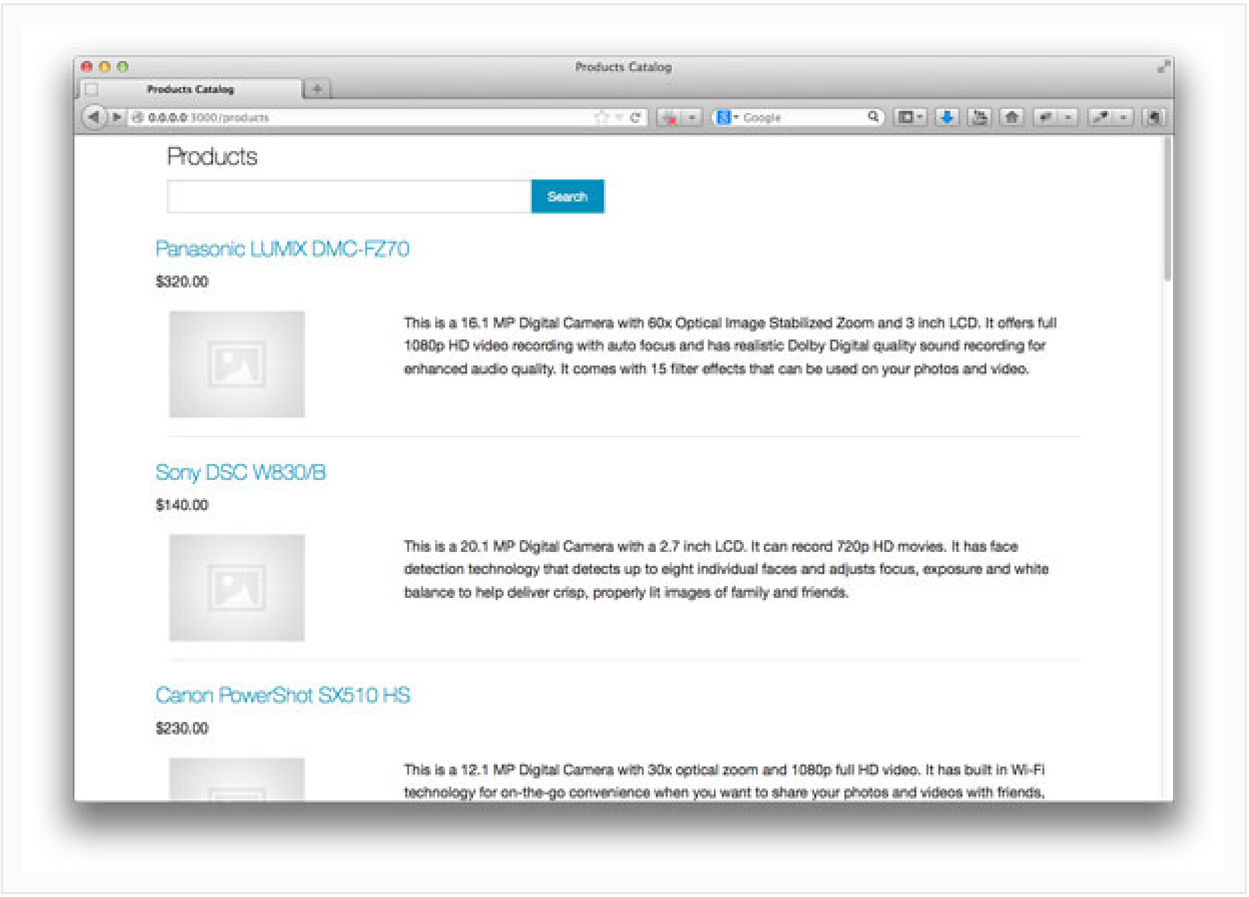
Full Text Search in Rails

by Joyce Echessa 28 Apr 2014

Difficulty: Intermediate Length: Medium Languages: English ▾

Web Development Ruby Ruby on Rails


Products Catalog
0.0.0.0:3000/products

Products

Search

Panasonic LUMIX DMC-FZ70
$320.00
This is a 16.1 MP Digital Camera with 60x Optical Image Stabilized Zoom and 3 inch LCD. It offers full 1080p HD video recording with auto focus and has realistic Dolby Digital quality sound recording for enhanced audio quality. It comes with 15 filter effects that can be used on your photos and video.

Sony DSC W830/B
$140.00
This is a 20.1 MP Digital Camera with a 2.7 inch LCD. It can record 720p HD movies. It has face detection technology that detects up to eight individual faces and adjusts focus, exposure and white balance to help deliver crisp, properly lit images of family and friends.

Canon PowerShot SX510 HS
$230.00
This is a 12.1 MP Digital Camera with 30x optical zoom and 1080p full HD video. It has built in Wi-Fi technology for on-the-go convenience when you want to share your photos and videos with friends.

What You'll Be Creating

Introduction

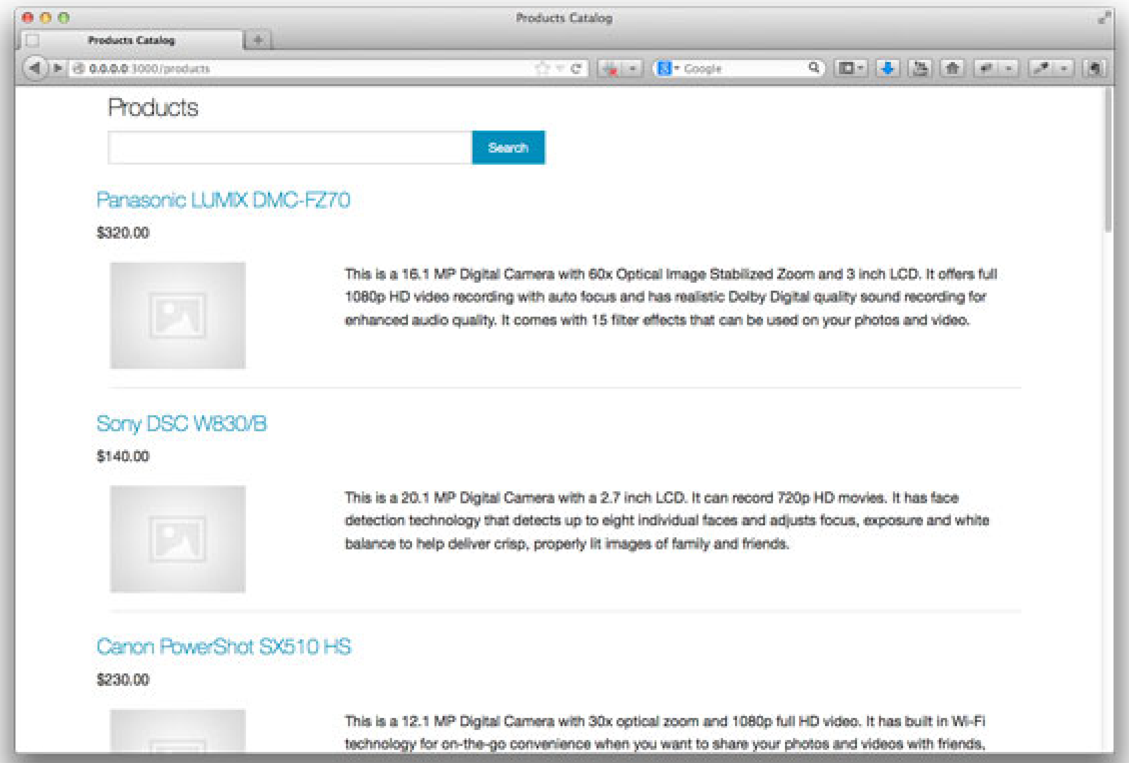
Searching records is a common requirement in web applications. There is usually a requirement to allow users to quickly access the data they want from large records. While it is possible to do this using simple SQL queries, sometimes it is more efficient to use a search engine.

Solr is a popular search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, near real-time indexing, dynamic clustering, database integration, rich document handling, and geospatial search. In this tutorial, we'll be looking at performing full text search using Sunspot, which is a library that enables integration of Solr in ruby applications.

Project Setup

I've created a simple app on Github which I'll be using here instead of starting with a new project. The app shows a list of products with their name, image, price and description. I have included some seed data so you can run `rake db:seed` if you don't want to input the data your self. The application uses Paperclip for image attachments and since I use image resizing, ImageMagick will need to be installed on your system. You'll also require the Java runtime installed on your machine to proceed with the tutorial.

The image below shows the application. The search form at the top does nothing at the moment, but we will enable a user to search through the products and get results based on not just the product name, but also on its description.



Products

Panasonic LUMIX DMC-FZ70

$320.00

This is a 16.1 MP Digital Camera with 60x Optical Image Stabilized Zoom and 3 inch LCD. It offers full 1080p HD video recording with auto focus and has realistic Dolby Digital quality sound recording for enhanced audio quality. It comes with 15 filter effects that can be used on your photos and video.

Sony DSC W830/B

$140.00

This is a 20.1 MP Digital Camera with a 2.7 inch LCD. It can record 720p HD movies. It has face detection technology that detects up to eight individual faces and adjusts focus, exposure and white balance to help deliver crisp, properly lit images of family and friends.

Canon PowerShot SX510 HS

$230.00

This is a 12.1 MP Digital Camera with 30x optical zoom and 1080p full HD video. It has built in Wi-Fi technology for on-the-go convenience when you want to share your photos and videos with friends,

Searching

We'll start off by including the Sunspot and Solr gems in our Gemfile. For development, we'll use the `sunspot_solr` gem that comes with a pre-packaged Solr distribution, therefore we won't need to install it separately.

```
gem 'sunspot_rails'

group :development do
    gem 'sunspot_solr'
end
```

Run `bundle install` and then run the following command to generate the Sunspot configuration file.

```
rails generate sunspot_rails:install
```

This creates the `/config/sunspot.yml` file which lets your app know where to find the Solr server.

To set up the objects that you want indexed, add a searchable block to the objects. In the starter project, we have a Product model with name, price, description and photo fields. We will enable a full-text search to be done on the name and description fields. In `/models/product.rb` add:

```ruby
searchable do
    text :name, :description
end
```

Start the Solr server by running:

```
rake sunspot:solr:start
```

Sunspot indexes new records that you create, but if you already have some records in the database, run `rake sunspot:reindex` to have them indexed.

We then add the code in the Products controller that will take the user's input and pass it to the search engine. In the code below, we call `search` on the Product model and pass in a block. We call the `fulltext` method in the block and pass in the query string that we want to be searched for. There are several methods we can use here to specify the search results we want. The search results are then assigned to `@products` which will be available to our view.

```ruby
def index
    @query = Product.search do
        fulltext params[:search]
    end
    @products = @query.results
end
```

Run the application and you should now be able to search through the available products.

Solr will do a case insensitive search through the product names and descriptions using the word or phrase input. You can make one field hold more weight than the other to improve the relevancy of your search results. This is done with the `boost` method which is passed a value that determines the priority assigned to the different fields. The field with the highest value will carry more importance.

In our application, we can specify the products which have the searched string in their name to be scored higher. We do this by making the following changes in

`/models/product.rb`.

```ruby
searchable do
    text :name, :boost => 2
    text :description
end
```

Reindex the records with `rake sunspot:reindex` and now the results with the searched term in the product name, will be placed higher than those with the term in the description. You can add more records to test this out.

Faceted Browsing

Faceted browsing is a way of navigating search data by way of various sets of associated attributes. For example, in our application, we can classify searches for products by price range and give counts of each range.

First add price to the `searchable` method in `/models/product.rb`

```ruby
searchable do
    text :name, :boost => 2
    text :description
    double :price
end
```

Then call `facet` in the controller. The products will be faceted by the range of their price in intervals of $100.00. Here we assume that all products cost less than $500.

```ruby
def index
    @query = Product.search do
        fulltext params[:search]

        facet :price, :range => 0..500, :range_interval => 100
        with(:price, Range.new(*params[:price_range].split("..").map(&:to_i))

    end
    @products = @query.results
end
```

In the view file, paste the following at the place you want to see the faceted results.

```erb
<div class="row">
    <h3>Search Results</h3>
    <ul>
        <% for row in @query.facet(:price).rows %>
            <li>
                <% if params[:price_range].blank? %>
```

```
                    <%= link_to row.value, :price_range => row.value, :search
                <% else %>
                    <%= row.value %> (<%= link_to "X", :price_range => nil %>
                <% end %>
            </li>
        <% end %>
    </ul>
</div>
```

Now when you search for a term, there will be a list of facets showing how many results are in each price range. In our example application, if you search for the word 'camera', you will see the following list.

```
100.0..200.0 (2)
200.0..300.0 (1)
300.0..400.0 (1)
```

Each item is a link and when clicked on, you will get a list of the products that meet your search term and that also fall into the price range you clicked on.

The link passes the original search query and the chosen range to the index action. Since it passes the range as a string, we use `Range.new(*params[:price_range].split("..").map(&:to_i))` to convert it back to a range. You could use conditional statements to output more user friendly links like `$100 - $199 (2)` instead of `100.0..200.0 (2)` but we won't get into that here.

Advanced Configurations

There are some further configurations you can do on Solr to customize how it works. In its default, Sunspot performs full-text search by dividing the search string into tokens based on whitespace and other delimiter characters using a smart tokenizer called the `StandardTokenizer`. Then the tokens are lower cased and the exact words are searched for.

This might be okay at times, but you might also want to configure the search engine to allow for human error or to allow queries to be made that aren't too strict. For instance, you might want to provide some synonyms to the engine so that when the user doesn't enter the exact text that is in your records, they might still find similar results. An example of this, is that you might have an item labeled 'ipod' in your records. You may provide synonyms like 'iPod', 'i-pod' and 'i pod' to increase the odds of users finding the data.

Another useful functionality you could add is stemming, which will allow Solr to match different words with the same root. For example, if the user entered 'run', they would get results with 'run' and 'running'. Or if they searched for 'walk', the results will include data that contains 'walk', 'walking', 'walked', and so on.

Solr settings are found in `solr/conf/schema.xml` and that is the file to modify to change the server's configuration. This is out of the scope of this tutorial, but for more on this, check out the advanced full-text config post and the Solr wiki.

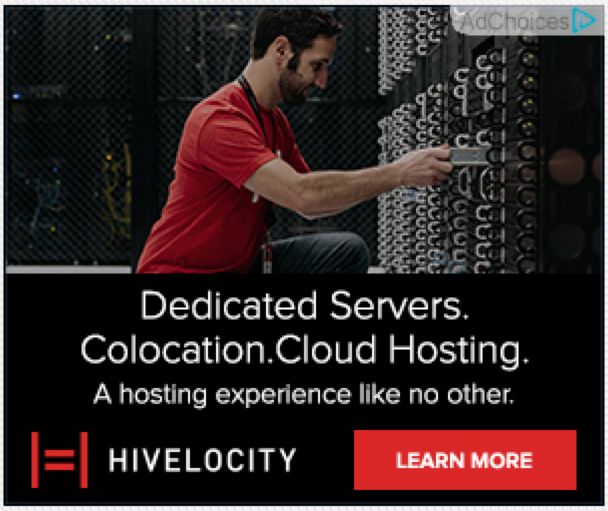
Conclusion

Now to finish up, stop the Solr server by running:

```
rake sunspot:solr:stop
```

We have looked at how to use the Sunspot gem to utilize the Solr search engine in a Rails app. Besides the settings we have used, there are plenty more you can use to customize your search results. Be sure to check the Readme file for more options.

Solr gives you the kind of searching capability that isn't easy to achieve with regular SQL queries. For simple apps, with a small amount of database records, SQL queries will do without much of a performance hit. But if you want something that is scalable, then it is worth looking into Solr or other available search engines.

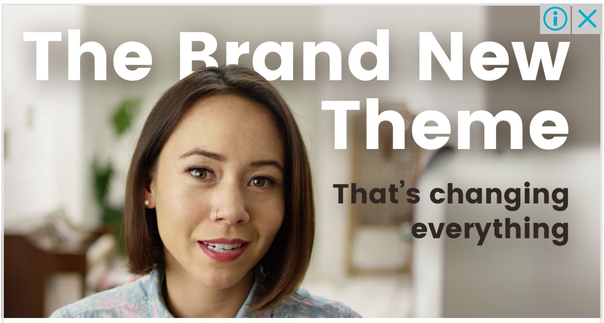

AdChoices
Dedicated Servers.
Colocation.Cloud Hosting.
A hosting experience like no other.
HIVELOCITY
LEARN MORE

Advertisement

Joyce Echessa

I am a web developer who dabbles in mobile development from time to time. You can find me on Twitter @joyceechessa or visit my website to see what I'm up to.

joyceechessa



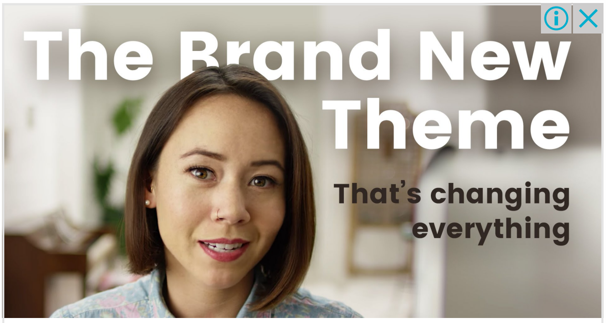
View Online Demo

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by native



8 Comments Tuts+ Hub Jeyavel

Recommend Share Sort by Best

Join the discussionâ€¦

Jeremy â€¢ 3 years ago

Please, Ruby code in general is 2 spaces indent. Except that, it was a good introduction of using Solr with Rails. Thanks

7 | â€¢ Reply â€¢ Share â€º

Tanveer Hussain â€¢ 3 years ago

Nice article but i think Elastic search is more batter then Solr.

1 | â€¢ Reply â€¢ Share â€º

brian piercy Tanveer Hussain â€¢ 3 years ago

Why?

| â€¢ Reply â€¢ Share â€º

Tanveer Hussain brian piercy â€¢ 3 years ago

Solr may be the weapon of choice when building standard search applications, but Elasticsearch takes it to the next level with an architecture for creating modern realtime search applications. Percolation is an exciting and innovative feature that singlehandedly blows Solr right out of the water. Elasticsearch is scalable, speedy and a dream to integrate with. Adios Solr.

| â€¢ Reply â€¢ Share â€º

Saworieza â€¢ 2 years ago

Had me going Yes! Yes! Yes! as always your tutorials are awesome.

| â€¢ Reply â€¢ Share â€º

venkata reddy â€¢ 3 years ago

configuring solr in production has always been a pain. we converted all our solr searching code to arel and the life became a lot easier now :)

| â€¢ Reply â€¢ Share â€º

Bharath â€¢ 3 years ago

Good one

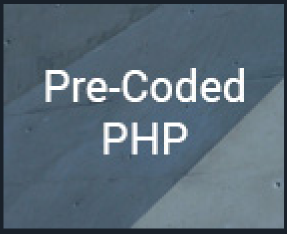
| â€¢ Reply â€¢ Share â€º

Josh bedo â€¢ 3 years ago

Awesome article definitely want to give this a go some time, my friend that runs his own search engine uses Solr for terabytes of data but I do tend to hear a lot about Elastic Search.

| â€¢ Reply â€¢ Share â€º

envatostudio
Expert freelancers for your project


Pre-Coded PHP

Studio's services

CodeCanyon