- **Ruby(https://www.sitepoint.com/ruby/)** **Article**

Full-Text Search in Rails with ElasticSearch

By Mostafa Abdulhamid (https://www.sitepoint.com/author/dosht/) August 11, 2014

In this article you will learn how to integrate ElasticSearch into a Rails application.

A full-text search engine examines all of the words in every stored document as it tries to match search criteria (text specified by a user) wikipedia (http://en.wikipedia.org/wiki/Full_text_search). For example, if you want to find articles that talk about Rails, you might search using the term “rails”. If you don’t have a special indexing technique, it means fully scanning all records to find matches, which will be extremely inefficient. One way to solve this is an “inverted index” that maps the words in the content of all records to its location in the database.

For example, if a primary key index is like this:

```
article#1 -> "breakthrough drug for schizophrenia"
article#2 -> "new schizophrenia drug"
article#3 -> "new approach for treatment of schizophrenia"
article#4 -> "new hopes for schizophrenia patients"
...
```

An inverted index for these records will be like this:

```
breakthrough    -> article#1
drug            -> article#1, article#2
schizophrenia   -> article#1, article#2, article#3, article#4
approach        -> article#3
new             -> article#2, article#3, article#4
hopes           -> article#4
...
```

Now, searching for the word “drug” uses the inverted index and return *article#1* and *article#2* directly.

I recommend the IR Book (http://www-nlp.stanford.edu/IR-book/), if you want to learn more about this.

Build an Articles App

We will start with the famous blog example (http://guides.rubyonrails.org/getting_started.html) used by the Rails guides.

Create the Rails App

Type the following at the command prompt:

```
$ rails new blog
$ cd blog
$ bundle install
$ rails s
```

Create the Articles Controller

Create the articles controller using the Rails generator, add routes to **config/routes.rb**, and add methods for showing, creating, and listing articles.

```
$ rails g controller articles
```

Then open **config/routes.rb** and add this resource:

```
Blog::Application.routes.draw do
  resources :articles
end
```

Now, open **app/controllers/articles_controller.rb** and add methods to create, view, and list articles.

```
 def index
  @articles = Article.all
end

def show
  @article = Article.find params[:id]
end

def new
end

def create
  @article = Article.new article_params
  if @article.save
    redirect_to @article
  else
    render 'new'
  end
end

private
  def article_params
    params.require(:article).permit :title, :text
  end
```

Article Model

We'll need a model for the articles, so generate it like so:

```
 $ rails g model Article title:string text:text
$ rake db:migrate
```

Views

New Article Form

Create a new file at **app/views/articles/new.html.erb** with the following content:

```
 <h1>New Article</h1>

<%= form_for :article, url: articles_path do |f| %>

  <% if not @article.nil? and @article.errors.any? %>
  <div id="error_explanation">
    <h2><%= pluralize(@article.errors.count, "error") %> prohibited
      this article from being saved:</h2>
    <ul>
    <% @article.errors.full_messages.each do |msg| %>
      <li><%= msg %></li>
    <% end %>
    </ul>
  </div>
  <% end %>

  <p>
    <%= f.label :title %><br>
    <%= f.text_field :title %>
  </p>

  <p>
    <%= f.label :text %><br>
    <%= f.text_area :text %>
  </p>

  <p>
    <%= f.submit %>
  </p>
<% end %>

<%= link_to '<- Back', articles_path %>
```

Show One Article

Create another file at **app/views/articles/show.html.erb**:

```
 <p>
   <strong>Title:</strong>
   <%= @article.title %>
 </p>

 <p>
   <strong>Text:</strong>
   <%= @article.text %>
 </p>

 <%= link_to '<- Back', articles_path %>
```

List All Articles

Create a third file at **app/views/articles/index.html.erb**:

```
 <h1>Articles</h1>

 <ul>
   <% @articles.each do |article| %>
     <li>
       <h3>
         <%= article.title %>
       </h3>
       <p>
         <%= article.text %>
       </p>
     </li>
   <% end -%>
 </ul>
 <%= link_to 'New Article', new_article_path %>
```

You can now add and view articles. Make sure you start the Rails server and go to http://localhost:3000/articles (http://localhost:3000/articles). Click on “New Article” and add a few articles. These will be used to test our full-text search capabilities.

Integrate ElasticSearch

Currently, we can find an article by *id* only. Integrating ElasticSearch will allow finding articles by any word in its title or text.

Install for Ubuntu and Mac

Ubuntu

Go to elasticsearch.org/download (http://www.elasticsearch.org/download/) and download the **DEB** file. Once the file is local, type:

```
$ sudo dpkg -i elasticsearch-[version].deb
```

Mac

If you're on a Mac, Homebrew makes it easy:

```
$ brew install elasticsearch
```

Validate Installation

Open this url: http://localhost:9200 (http://localhost:9200) and you'll see ElasticSearch respond like so:

```
{
  "status" : 200,
  "name" : "Anvil",
  "version" : {
    "number" : "1.2.1",
    "build_hash" : "6c95b759f9e7ef0f8e17f77d850da43ce8a4b364",
    "build_timestamp" : "2014-06-03T15:02:52Z",
    "build_snapshot" : false,
    "lucene_version" : "4.8"
  },
  "tagline" : "You Know, for Search"
}
```

Add Basic Search

Create a controller called `search`, along with a view so you can do something like: */search?q=ruby*.

Gemfile

```
gem 'elasticsearch-model'
gem 'elasticsearch-rails'
```

Remember to run `bundle install` to install these gems.

Search Controller

Create The `SearchController`:

```
$ rails g controller search
```

Add this method to **app/controller/search_controller.rb**:

```
def search
  if params[:q].nil?
    @articles = []
  else
    @articles = Article.search params[:q]
  end
end
```

Integrate Search into Article

To add the ElasticSearch integration to the Article model, require `elasticsearch/model` and include the main module in `Article` class.

Modify **app/models/article.rb**:

```
require 'elasticsearch/model'

class Article < ActiveRecord::Base
  include Elasticsearch::Model
  include Elasticsearch::Model::Callbacks
end
Article.import # for auto sync model with elastic search
```

Search View

Create a new file at **app/views/search/search.html.erb**:

```
<h1>Articles Search</h1>

<%= form_for search_path, method: :get do |f| %>
  <p>
    <%= f.label "Search for" %>
    <%= text_field_tag :q, params[:q] %>
    <%= submit_tag "Go", name: nil %>
  </p>
<% end %>

<ul>
  <% @articles.each do |article| %>
    <li>
      <h3>
        <%= link_to article.title, controller: "articles", action: "show",
      </h3>
    </li>
  <% end %>
</ul>
```

Search Route

Add the search route to _*config/routes.rb*-:

```
get 'search', to: 'search#search'
```

You can now go to http://localhost:3000/search (http://localhost:3000/search) and search for any word in the articles you created.

Enhance the Search

You may notice that there are some limitations in your search engine. For example, searching for part of a word, such as *"rub"* or *"roby"* instead of *"ruby"*, will give you zero results. Also, it'd be nice if the search engine gave results that include words similar to your search term.

ElasticSearch provides a lot of features to enhance your search. I will give some examples.

Custom Query

There are different types of queries that we can use. So far, we are just using the default ElasticSearch query. To enhance search results, we need to modify this default query. We can, for example, give higher priority for fields like *title* over other fields.

ElasticSearch provides a full Query DSL based on JSON to define queries. In general, there are basic queries, such as term or prefix. There are also compound queries, like the bool query. Queries can also have filters associated with them, such as the filtered or constant_score queries.

Let's add a custom search method to our article model in **app/models/article.rb**:

```
def self.search(query)
  __elasticsearch__.search(
    {
      query: {
        multi_match: {
          query: query,
          fields: ['title^10', 'text']
        }
      }
    }
  )
end
```

Note: ^10 boosts by 10 the score of hits when the search term is matched in the title

Custom Mapping

Mapping is the process of defining how a document should be mapped to the Search Engine, including its searchable characteristics like which fields are searchable and if/how they are tokenized.

Explicit mapping is defined on an index/type level. By default, there isn't a need to define an explicit mapping, since one is automatically created and registered when a new type or new field is introduced (with no performance overhead) and has sensible defaults. Only when the defaults need to be overridden must a mapping definition be provided.

We will improve the search so that you can search for a term like *"search"* and receive results also including *"searches"* and *"searching"* ..etc. This will use the built-in English analyzer in ElasticSearch to apply word stemming (http://xapian.org/docs/stemming.html) before indexing.

Add this mapping to the Article class: at **app/models/article.rb**

```
settings index: { number_of_shards: 1 } do
  mappings dynamic: 'false' do
    indexes :title, analyzer: 'english'
    indexes :text, analyzer: 'english'
  end
end
```

It's a good idea to add the following lines to the end of the file to automatically drop and rebuiled the index when **article.rb** is loaded:

```
# Delete the previous articles index in Elasticsearch
Article.__elasticsearch__.client.indices.delete index: Article.index_name r

# Create the new index with the new mapping
Article.__elasticsearch__.client.indices.create \
  index: Article.index_name,
  body: { settings: Article.settings.to_hash, mappings: Article.mappings.to

# Index all article records from the DB to Elasticsearch
Article.import
```

Search Highlighting

Basically, we'd like to show the parts of the articles where the term we are searching for appears. It's like when you search in google and you see a sample of the document that includes your term in bold. In ElasticSearch, this is called "highlights". We will add a *highlight* parameter to our query and specify the fields we want to highlight. ElasticSearch will return the term between an tag, along with a few words before and after the term.

Assuming we are searching for the term *"opensource"*, ElasticSearch will return something like this:

```
Elasticsearch is a flexible and powerful <em>opensource</em>, distributed,
```

Note that *"opensource"* is surounded by an tag.

Add Highlights to the SearchController

First, add the “highlight” parameter to the ElasticSearch query:

```
def self.search(query)
  __elasticsearch__.search(
    {
      query: {
        multi_match: {
          query: query,
          fields: ['title^10', 'text']
        }
      },
      highlight: {
        pre_tags: ['<em>'],
        post_tags: ['</em>'],
        fields: {
          title: {},
          text: {}
        }
      }
    }
  )
end
```

Show Highlights in View

It’s pretty easy show this highlight in the view. Go to **app/views/search/search.html.erb** and replace the `ul` element with this:

```
<ul>
  <% @articles.each do |article| %>
    <li>
      <h3>
        <%= link_to article.try(:highlight).try(:title) ? article.highligh
          controller: "articles",
          action: "show",
          id: article._id%>
      </h3>
      <% if article.try(:highlight).try(:text) %>
        <% article.highlight.text.each do |snippet| %>
          <p><%= snippet.html_safe %>...</p>
        <% end %>
      <% end %>
    </li>
  <% end %>
</ul>
```

Now add a style for in **app/assets/stylesheets/search.css.scss**:

```
em {
  background: yellow;
}
```

One last thing we need is the highlighted term returned by ElasticSearch to be surrounded by a few words. If you need to show the title from the beginning, add `index_options: 'offsets'` to the title mapping:

```
settings index: { number_of_shards: 1 } do
  mappings dynamic: 'false' do
    indexes :title, analyzer: 'english', index_options: 'offsets'
    indexes :text, analyzer: 'english'
  end
end
```

This was a quick example for integerating ElasticSearch into a Rails app. We added basic search, then mixed things up a little using custom queries, mapping, and highlights. You can download the full source from here (https://github.com/dosht/rails-elasticsearch-example)

References

- elasticsearch/elasticsearch-rails (https://github.com/elasticsearch/elasticsearch-rails)
- rails-application-templates (https://github.com/elasticsearch/elasticsearch-rails/tree/master/elasticsearch-rails#rails-application-templates)
- Mapping Guide (http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/mapping.html)
- Query-DSL Guide (http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/query-dsl.html)
- rubyonrails.org/getting_started.html (http://guides.rubyonrails.org/getting_started.html)

Was this helpful?

Meet the author

Mostafa Abdulhamid (https://www.sitepoint.com/author/dosht/)

22 Comments SitePoint Jeyavel

Recommend 1 Share Sort by Best

Join the discussionâ€¦

Andrea Rocca â€¢ 18 days ago

Awesome tutorial, thanks! For some reason, I have followed the whole thing but when I load my view I get the error: cannot load such file -- elasticsearch/model. Any idea?

I have included both the gems in my gemfile and I made sure they are installed

â€¢ Reply â€¢ Share â€º

Tobi â€¢ 4 months ago

Greate Tutorial!.It worked as well . Can you help me anwser 2 below problems. Thanks you.
1. I opened elasticsearch log. But i cant saw anything. It not create new log after search ?. I want to config to see log after search action.
2. How can i scale full-text-search for mutil models.

â€¢ Reply â€¢ Share â€º

Matt Welke â€¢ 5 months ago

Thanks for creating this tutorial. I was wondering if you could help me with an issue I ran into. I followed up until I had finished creating my search controller and view, and was running the application. The "articles" routes all result in this error. The "search" route renders, but upon clicking "OK" after entering an ES search term, I get the error too:

ArgumentError in SearchController#search
articles does not exist to be imported into. Use create_index! or the :force option to create it.

Thanks

^ | v â€¢ Reply â€¢ Share â€º

Muhamad Akbar Bin Widayat â€¢ a year ago

Thanks a lot for this article. It helps me a lot. :)

^ | v â€¢ Reply â€¢ Share â€º

Andrey â€¢ a year ago

<% if not @article.nil? and @article.errors.any? %>
Why did you use "not" and "and" instead of "!" and "||"?

^ | v â€¢ Reply â€¢ Share â€º

Abhimanyu Aryan â€¢ a year ago

I wish to write my search algorithms for my Rails app from scratch. SitePoint can you provide me links which could help me?

I know searching algorithms though :)

^ | v â€¢ Reply â€¢ Share â€º

altuzar â€¢ a year ago

Thanks a lot for this! Works great! You saved my life! \o/

^ | v â€¢ Reply â€¢ Share â€º

MR L â€¢ a year ago

At first thanks for this article,

it is great, i help me a lot, of course thanks to @karmi too, with his contributions to elastic on github.

i only want to say , at the post, there is a litlle mistake , it is not
"Add Highlights to the SearchController" the highligths must add to model article, in fact whole def self.search method.

once again @dosht thaks.

^ | v â€¢ Reply â€¢ Share â€º

Mo3G â€¢ 2 years ago

Thank's a lot for this tutorial !

^ | v â€¢ Reply â€¢ Share â€º

rrrub â€¢ 2 years ago

Quick question here, how do you show results into a different template?

^ | v â€¢ Reply â€¢ Share â€º

Benoit â€¢ 2 years ago

ok, I have read a little and finally have understood...I don't have to use multiple index but just multiple analyzer.
Does anyone know how to set __elasticsearch__.search with analyzer = "my_custom_analyzer" ??

â€¢ Reply â€¢ Share â€º

Benoit â€¢ 2 years ago

Hello !

Thak you very much for this, clear, short, and excellent article. Thanks to you, I have setup the ES search within my rail app :-) !

Just a question. In your example, let's say I have 2 very different __elasticsearch.search functions (one cutomized for typeahead with ngram, the other for a basic search...). My idea was to build 2 different indices and to do "function search1 on index 1", "function search2 on index 2".

Is it possible and not too much a bad strategy ?

How can I delete /create/refresh 2 different indices in a same model ?

How can I specify in the query, which index a function should be executed on ?

You can find my related question on stackoverflow:

http://stackoverflow.com/quest...

Thank you very much! Best regards !

Benoît

â€¢ Reply â€¢ Share â€º

babaleo â€¢ 2 years ago

Great article! This has helped me quickly integrate ES into my Rails app.

â€¢ Reply â€¢ Share â€º

Mohamed Fadlalla â€¢ 2 years ago

Nice Topic, Thank you for this

â€¢ Reply â€¢ Share â€º

Anthony Candaele â€¢ 2 years ago

I just deployed my Rails app, with the ElasticSearch implementation to Heroku, but the app is crashing. I added the Heroku add-on Bonsay-ElasticSearch. Am I missing something? My repository is at: https://github.com/acandael/po...

â€¢ Reply â€¢ Share â€º

Anthony Candaele â€¢ 2 years ago

Excellent article, this was just what I need. Just successfully implement ElasticSearch in my Rails app, with the help of this blog post. One question though, in the blog post the search is only implemented for articles. In my application though I need to do search on more then one model. I need to implement search for 'Publication' 'ResearchProject' and 'Researcher'. How can I implement searches on multiple models?

â€¢ Reply â€¢ Share â€º

Adam Hegyi Anthony Candaele â€¢ 2 years ago

Adam Hegyi ➦ Anthony Candaele â€¢ 2 years ago

As far as I know es-rails doesn't support multi index search. You can try creating one index for all the models. So index_name would be the same for all models and also the mapping. It works quite well if your models has similar attributes.

^ | ˅ â€¢ Reply â€¢ Share â€º

Anthony Candaele ➦ Adam Hegyi â€¢ 2 years ago

Thanks Adam, In the meantime I just added an instance variable for every model to my SearchController:

```
class SearchController < ApplicationController
def search
@search_term = params[:q]
if params[:q].nil?
@publications = []
@research_projects = []
else
@publications = Publication.search params[:q]
@research_projects = ResearchProject.search params[:q]
end
end
end
```

It may not be the most optimized way, but it works. I think the 'one index' approach wouldn't work, as my models have different attributes.

Thanks for your suggestion,

Anthony

^ | ˅ â€¢ Reply â€¢ Share â€º

Adam Hegyi ➦ Anthony Candaele â€¢ 2 years ago

one index might work, just your mapping would be bigger:

```
{
publication: {

# publication_attributes if your model.is_a? Publication
},
research_project: {
# research_project_attributes if your model.is_a? ResearchProject
}
...other models
}
```

If your document is a publication then basically research_project would be empty.

This is also not the best solution but with this you would have only one result set. Works well if you want to paginate your results.3

^ | ˅ â€¢ Reply â€¢ Share â€º

Abraham ➦ Adam Hegyi â€¢ 2 years ago

See this thread for searching across all indices: https://github.com/elastic/ela...

^ | ˅ â€¢ Reply â€¢ Share â€º

karmi â€¢ 2 years ago

Thanks for the article! Note that you can rebuild the index with a single command: `Article.import force: true` -- or use the corresponding environment variable for the Rake task.

â€¢ Reply â€¢ Share â€º

Anthony Candaele â€¢ 2 years ago

I deployed my ElasticSearch implementation to Heroku, but the app crashes. Installing the Bonsai-ElasticSearch add-on didn't solve the issue. The error message in the Heroku logs is:

```
from bin/rails:8:in `require'

2014-08-14T12:51:12.352016+00:00 app[web.1]: from bin/rails:8:in `<main>'

2014-08-14T12:51:13.470645+00:00 heroku[web.1]: State changed from starting to crashed

2014-08-14T12:51:13.447331+00:00 heroku[web.1]: Process exited with status 1

2014-08-14T13:25:30.971800+00:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET path="/" host=morning-anchorage-2469.herokua... request_id=353c4170-531c-4ff1-9afd-4975169e5176 fwd="81.245.163.54" dyno= connect= service= status=503 bytes=

2014-08-14T13:25:31.401546+00:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET path="/favicon.ico" host=morning-anchorage-2469.herokua... request_id=a42774ab-930a-4c8b-81d4-bf1c91617f1e fwd="81.245.163.54" dyno= connect= service= status=503 bytes=

2014-08-14T13:32:01.917187+00:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET path="/" host=morning-anchorage-2469.herokua... request_id=66c4cbfb-79ef-4d97-b297-b50d2595b284 fwd="81.245.163.54" dyno= connect= service= status=503 bytes=
```

and Developers

LATEST BOOKS › (/premium/books/)

PREMIUM BOOK

Rails: Novice to Ninja (https://www.sitepoint.com/premium/books/rails-novice-to-ninja)

PREMIUM BOOK

Jump Start Git (https://www.sitepoint.com/premium/books/jump-start-git)

PREMIUM BOOK

Jump Start Rails (https://www.sitepoint.com/premium/books/jump-start-rails)

Get the latest in Ruby, once a week, for free.

Enter your email

Subscribe

Facebook LinkedIn Twitter