Solr

Your very own search engine!

What’s this Lucene thing?

- Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform ones.
- This type of index is called an inverted index, because it inverts a page-centric data structure (page->words) to a keyword-centric data structure (word->pages).
- **Important: instead of searching the text directly, Lucene searches an index instead.**

What is Solr?

- At it’s most basic it’s a search server built on top off Apache Lucene that provides an array of features.
- On the website: “highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more”
- How I’ve used it: a simple and high performance way to do database searches
- Reference manual is over 500 pages.

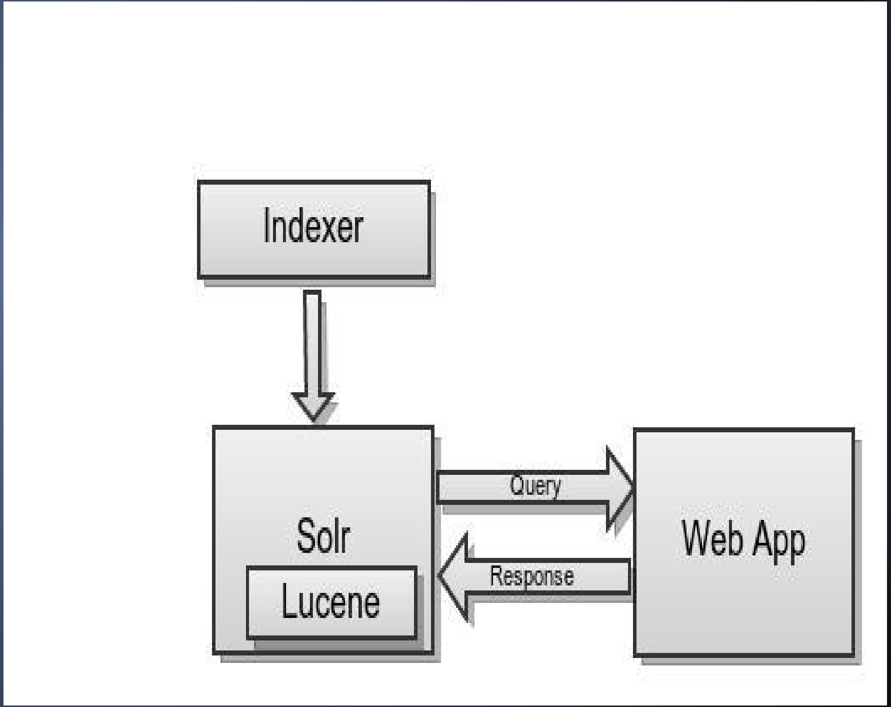
How does Solr work?

A. Solr Searches the Indexed Documents.

B. Each Document has an ID and list of terms

C. For Each term, solr has lists of all document that contains this specific term.

D. Identify each document by it’s ID and return it.

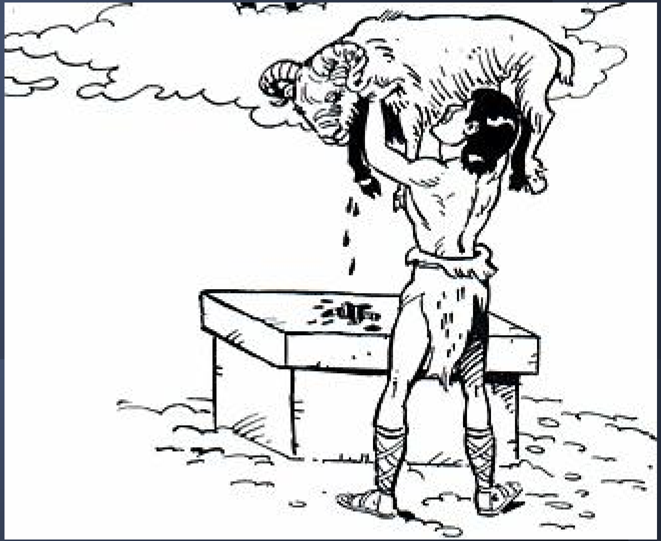


TL;DR - Solr is a Lucene Frontend that provides fast text search on a DB.

Now onto the fun stuff!

Lets see it in action...

(A sacrifice to the live action Gods)



A real use case:

CU FCQ - A Ruby on Rails project to visualize CU’s FCQs.

- Dirty scraped data.
- Maintain a DB with over 100,000 rows.
- Needs to be searchable.
- Needs to be fast.

What I was doing before:

```ruby
def self.search(query)
  #If there wasn't any input return everything.
  if query.nil?
    return Instructors.all
  end
  #this makes capital and strips whitespace
  q =  query.upcase.strip
  #actually make an sql query string using the input
  if q.include?(" ")
    where("instructor_last like ? AND instructor_first like ? OR instructor_first
like ? AND instructor_last like ?",q.split[1],q.split[0],q.split[1],q.split[0])
  else
    where("instructor_last = ? OR instructor_first = ? ",q,q)
  end
end
```

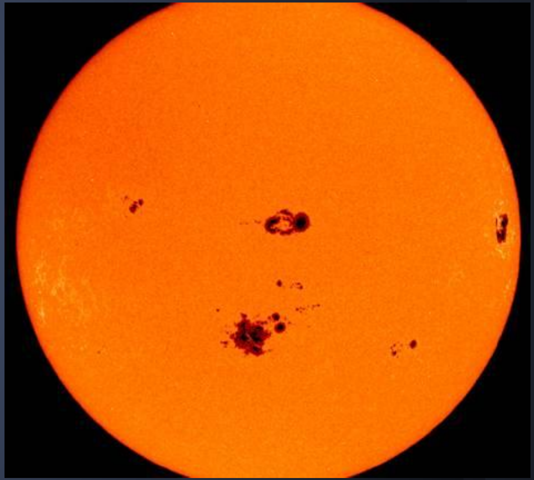
Notes:

1. Don’t laugh.
2. This was in my controller.
3. I was taking input directly from an HTML input box.
4. Use the input to create a SQL string and search.

This is bad on a number of levels.

Enter Sunspot

- Where to begin: I Googled “Search Engine in Rails” and came up with this gem.
  - A gem is a little library that you can download to plugin into your app.
- Sunspot is built on top of the RSolr library, which provides a low-level interface for Solr interaction; Sunspot provides a simple, intuitive, expressive DSL backed by powerful features for indexing objects and searching for them.
- **TL;DR Comes with a bundled version of Solr, so boilerplate is good to go out of the box.**



Integrating it into my project (pt. 1)

Models (Our Logic):

- Add the searchable fields for the classes.

```
Instructor.rb-
  searchable do
    text :instructor_first
    text :instructor_last, :default_boost => 2
  end
Course.rb-
  searchable do
    text :crse
    text :subject
    text :course_title, :default_boost => 5
  end
```

Controller (DB <-> View):

- When we are calling an index for instructors we can perform our search.
- We also get pagination (30 by default)

```
def index
   @search = Instructor.search do
     fulltext params[:search]
     paginate :page => params[:page] || 1
   end
   @instructors = @search.results
 end
```

Displaying it to user (pt. 2)

Our Instructors Index view:

- This is where we list the return values of our query

```
<th>Instructor</th>
<th>Instructor Type</th>
      ...
<th>Last FCQ</th>

<tbody>
  <% @instructors.each do |instructor| %>
  <td><%= link_to instructor.name, instructor %
></td>
  <td><%= instructor.instr_group %></td>
               ....
  <td><%= instructor.latest_teaching %></td>
</tbody>
   <% end %>
<%= will_paginate @instructors %>
```

The Search Box :

- We add the search box so the user can interact with Solr.
- Anything with `<%= … %>` is Ruby code.
- Note: I removed indentation so it's more readable.

```
<form class="navbar-form navbar-right">

  <%= form_tag(instructors_path, :method => "get", id: "form-inline")
do %>

  <%= text_field_tag :search, params[:search], placeholder: "Search"
%>

   <%= submit_tag "Search", name: nil, :class=>'button' %>

   <% end %>
```

Putting it all together

- At this point, simply populate your DB again. (Pitfall 0)
- Start Solr and reindex everything
  - Note: Every time you need to change what Solr is indexing (new terms, new weights, etc.), you need to reindex.
- Start your web server and search!

Pitfalls

Pitfall 0: Solr and Rake

- To initialize our DB, we need to make sure Solr is killed.
- We want to remove all of our old temporary information and indices.
- Before we start actually putting Data into the DB, we restart our Solr server.
- Run our rake tasks.
- Reindex Solr once our DB is full.

```bash
#!/bin/bash
#kill solr process
pkill -f solr
#remove old data
rm -rf solr/data
rm -rf solr/default
rm -rf solr/development
rm -rf solr/pids
rm -rf solr/test
#startup solr in development environment
rake sunspot:solr:start RAILS_ENV=development
#all of our rake tasks (yours will be different)
bundle exec rake db:reset
...
bundle exec rake grades
#reindex solr
RAILS_ENV=development bundle exec rake sunspot:solr:
reindex
```

Pitfall 1: Starting your Server w/ Solr

- Just starting Sunspot can be a challenge.
- You need to specify what type of server you are going to be using with Sunspot (development)
- Each of these listen on different ports, so you might get "Connection Refused" or "404" errors if Sunspot can't connect to Solr.
- This script starts and reindexes Solr along with our web server.

```
#!/bin/bash
echo "Starting/Reindexing Solr"


rake sunspot:solr:start RAILS_ENV=development
rake sunspot:solr:reindex RAILS_ENV=development


echo "Solr successful! Starting Rails"


rails server -b 0.0.0.0 -p 80
echo "rails server closed!"
```

Pitfall 2: Solr configs and Git

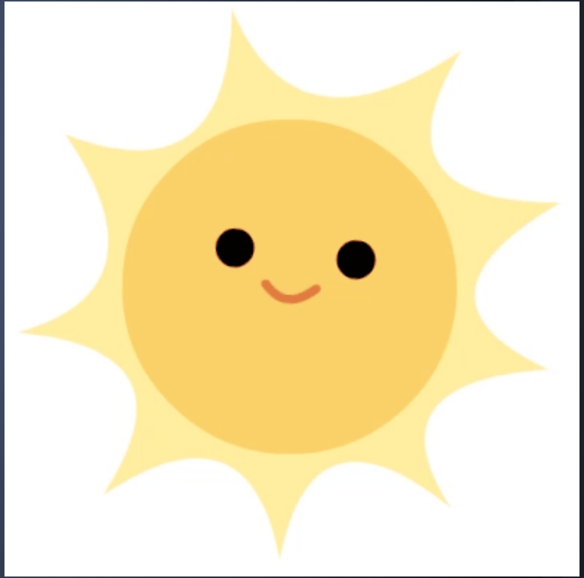
- You want to be able share your project on Git.
- The `solr/` directory contains import config information and temporary data (pid,index,etc.)
- Add the following code to your .gitignore so you don't start tracking any temp data that could wreck clones.
- You are still tracking import config information.
- Only keep `conf/` and solr.xml

```
#Ignore all of the solr stuff except the xml and the conf
solr/data
solr/default
solr/development
solr/pids
solr/test
```

Conclusion

- Solr is a super cool and (relatively) simple way to get search functionality for your DB.
- It’s fast, reliable, and has fantastic features like pagination and indexing.
- Interested in CUFCQ?
  - This is our Alpha launch and we’re looking for help.

You can access us @ **cufcq.com**



Important/Interesting Links:

- Integrating Rails with Solr tutorial.
- Basic explanation of Solr.
- Solr Sunspot.
- Our Project.

Need help? Email me (Alex Tsankov) at:

antsankov@gmail.com