Thinking Sphinx

Overview

Thinking-sphinx is a Ruby library which connects ActiveRecord to the Sphinx search daemon, managing the configuration and searching. With Thinking Sphinx models are linked up to sphinx easily, making searches across multiple fields much easier than using SQL.

What is Sphinx

Sphinx is a search engine and like many others it is fed documents, each with a unique identifier and text, and sphinx will index them. Then search terms can be passed and sphinx will return the most relevant documents.

The sphinx daemon (searchd) talks to a collection of indexes. Each index tracks a set of documents and each document is made up of fields and attributes.

* Fields: Content for the search queries in string form. For a model if specific words want to be searched, they must be in a fields of the index.
* Attributes: Used for sorting, filtering, and grouping search results. Attribute values do not get paided any attention by sphinx for search terms, however the data type can be: intergers, floats, datetimes, booleans, and strings.
* Multi-value attributes: Sphinx can handle arrays of attributes for a single document

Install

First install the sphinx daemon, by downloading the most recent binaries and placing them in /usr/local/bin. Modern versions of sphinx come with mysql and psql support built in.

Thinking Sphinx can be installed using ruby gems:

gem 'mysql2', '~> 0.4.10', :platform => :ruby

gem 'thinking-sphinx', '~> 4.0'

Note that the since sphinx communicates through the MySQL protocol the 'mysql2' gem is required even if the database being used is not MySQL.

Thinking Sphinx will create a new folder /db/sphinx and config files /config/ENV.sphinx.conf which should be added to the .gitignore of the project.

Indexing

The app/indices directory is used for setting up the indices for models, with the naming format of:

model\_name\_index.rb

Inside the index file, the following syntax is used and thinking sphix will translate the definitions into Sphinx configuration:

ThinkingSphinx::Index.define :article, :with => :active\_record do

indexes subject, :sortable => true

indexes content

indexes author.name, :as => :author, :sortable => true

has author\_id, created\_at, updated\_at

end

Syntax:

The model and the active record processor are defined to allow for SQL backed indices. Real-time indices can be used instead of SQL-backed indices which provide better indexing if data is changing frequently and it needs to be kept upto date. While real-time indicies update sphinx directly, they can be slower than SQL backed since they have to iterate through each record separtely. Real-time indices also support ruby methods instead of directly relying on model columns.

# for real-time indices:

ThinkingSphinx::Index.define :article, :with => :real\_time do

# ...

# for SQL-backed indices:

ThinkingSphinx::Index.define :article, :with => :active\_record do

# ...

Fields are set using:

indexes <model-column-name>

Options:

* renamed using ':as => :<new-name>'
* sorted using ':sortable => true'

Attributes are set using:

has <model-column-name>

Custom condtions and groups can be set using:

where <sql-query>

group\_by <sql-query>

Process Data

Once Sphinx is set up, run the rake commands to start sphinx and index the models:

bundle exec rake ts:start

bundle exec rake ts:index

If any changes are made to sphinx, the configuration must be rebuild using:

bundle exec rake ts:rebuild

Searching

Thinking Sphinx adds a search method to the indexed models:

Article.search('pancakes')

If searching using user input, make sure to sanitise inputs then escape any charaters using:

Article.search(ThinkingSphinx::Query.escape(params[:query]))

Queries can be focused on a specific field using :conditions option:

Article.search :conditions => {:subject => 'pancakes'}

Queries can be filtered by attributes using the :with option:

Article.search 'pancakes', :with => {:author\_id => @pat.id}

Article.search 'pancakes', :without => {:author\_id => @pat.id}

Article.search 'pancakes', :with\_all => {:tag\_ids => @tags.collect(&:id)}

Article.search 'pancakes', :with\_all => {:tag\_ids => [[1,2], 3]}

these filters can accept arrays and ranges, where as fields can only accept strings

It is also possible to search all indexed models in the application using:

ThinkingSphinx.search 'pancakes'

ThinkingSphinx.search 'pancakes', :classes => [Article, Comment]

ThinkingSphinx.search 'pancakes', :indices => ['article\_core']

Queries can be sorted by values:

Article.search "pancakes", :order => :created\_at

Nil results can end in errors, therefore thinking-sphinx can retry searches when nil results are found if the retry stale tag is passed:

Article.search 'pancakes', :retry\_stale => true

Pagination

Sphinx allways paginates searches, however it is possible to control the size of pages:

Article.search 'pancakes', :page => params[:page], :per\_page => 42

Search Results

By default thinking-sphinx returns ActiveRecord objects as search results, however it is possible to get simply the object ids using:

Article.search\_for\_ids 'pancakes'

There are several methods which can be used on search results:

@articles.total\_entries

@articles.total\_pages

@articles.current\_page

@articles.per\_page

Search results can be grouped by query:

Post.search 'syrup', :group\_by => :category\_id,

groups can be ordered by count and other sql queries:

Post.search 'syrup', :group\_by => :category\_id, :order\_group\_by => 'count(\*) desc'

groups can then be iterated over:

posts.each\_with\_group { |post, group| }

posts.each\_with\_count { |post, count| }

posts.each\_with\_group\_and\_count { |post, group, count| }

Catching Exceptions

By default, Thinking Sphinx does not run search queries until the search result is examined. This allows for scopes to be chaine without sending mutliple queries to Sphinx. However, it does also mean that exceptions will be raised in the view of the models, where error catching may not be present.

To force Thinking Sphinx to run seach queries when they are defined, use the populate key:

Article.search 'pancakes', :populate => true

Scopes

While scopes can be added to searches, they will not affect the seach results, just the SQL queries used to collect the ActiveRecord objects. However, scopes can be added to Thinking Sphinx which work in a similar manner.

Scopes are added in the model:

class Article < ActiveRecord::Base

include ThinkingSphinx::Scopes

sphinx\_scope(:latest\_first) {

{:order => 'created\_at DESC, @relevance DESC'}

}

sphinx\_scope(:by\_name) { |name|

{:conditions => {:name => name}}

}

# ...

end

Similar to ActiveRecord scopes they can just be called after the model being seached:

@articles = Article.latest\_first.by\_name('Puck').search 'pancakes'

Delta Indexes

When using SQL-backed indices, keeping records up to date can be limited in Sphinx due to records only be able to be updated by reprocessing the indices they're stored in. Therefore, to avoid this issue, the delta index can be used to only track changed documents, and because it is smaller it is much faster to process.

To add to sphinx, a delta column must be added to indexed models:

def self.up

add\_column :articles, :delta, :boolean, :default => true, :null => false

add\_index :articles, :delta

end

ThinkingSphinx::Index.define :article, :with => :active\_record,

:delta => true do

# ...

end

Then rebuild the sphinx

rake ts:rebuild

One issue with delta indexing is that on busy websites it creates a noticable speed decrease if performed inline with the request response cycle. Therefore, to avoid this problem, delta indexing should be passed off to a background worker such as delayedjob or sidekiq. Since the background process will need write access to sphinx, the worker must be based on the same machine as the sphinx deamon.

To set this up in the application add a background worker gem, then the relevant Thinking Sphinx integration gem (such as ts-sidekiq-delta). Then add the delayed worker in the delta key of the model index definition:

ThinkingSphinx::Index.define(:book, :with => :active\_record, :delta => ThinkingSphinx::Deltas::SidekiqDelta) do

# ...

end

To keep delta indexes quick, the delta must be merged into the core regularly. To do this, perform the task:

bundle exec rake ts:merge

This can be setup on a cronjob for the application.