Elasticsearch and Rails

Elasticsearch requires at least Java 8, please check your Java version first by running (and then install/upgrade accordingly if needed):

```
java -version
```

Install Java with brew, to verify which version it will install.

```
brew cask info java
```

Install java

```
brew cask install java
```

Or you can install Java from the homepage
https://www.java.com/en/download/help/download_options.xml

Download and unzip Elasticsearch.

Run

```
bin/elasticsearch
```

in console from the elasticsearch folder.

Add to the Gemfile

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

Bundle Install. First we add elasticsearch to the item.rb with code below

```
include Elasticsearch::Model
include Elasticsearch::Model::Callbacks
```

Then we create the indexing of the Item. We use a Analyzer to give to especific columns especific settings. In our case we want to handle the lowercase and we want to use a ngram tokenizer to split the words and look for part of the words. In the code below we see that we are using two different tokenizer. One produces N-grams with the min length of 3 amd max 5, and the other min length 2 and max 5.

```
  item_es_settings = {
  index: {
    analysis: {
      analyzer: {
        my_analyzer: {
          tokenizer: "my_tokenizer",
          filter: ["lowercase"]
        },
        my_second_analyzer: {
          tokenizer: "my_second_tokenizer",
          filter: ["lowercase"]
        }
      },
      tokenizer:{
        my_tokenizer: {
          type: "ngram",
          min_gram: 3,
          max_gram: 5,
          token_chars: [
            "letter",
            "digit"
          ]
        },
        my_second_tokenizer: {
          type: "ngram",
          min_gram: 2,
          max_gram: 5,
          token_chars: [
            "letter",
            "digit"
          ]
        }
      }
    }
  }
}
```

With the code below we do the indexing for the Item class. If we are using a column in the query we need the column in the index document. Like if we are using in the query :active_for_ar = true, we need in the document this column. As we can see in different columns we use the analyzer to split the words. For the indexing we give the type of the column. If we do not give the type Elasticsearch will use dynamic mapping to try to guess the field type.

```
settings item_es_settings do
    mappings dynamic: false do
        indexes :id, type: :text, analyzer: 'my_second_analyzer'
        indexes :name, type: :text, analyzer: 'my_analyzer', boost: 2
        indexes :group_notes, type: :text, analyzer: 'my_analyzer'
        indexes :manufacturer_name, type: :text
        indexes :group_name, type: :text, analyzer: 'my_analyzer', boost: 2
        indexes :category_name, type: :text
        indexes :product_line_name, type: :text
        indexes :product_line_sequence, type: :text
```

```
        indexes :group_id, type: :text, analyzer: 'my_second_analyzer'
        indexes :published_for_web, type: :boolean
        indexes :active_for_ar, type: :boolean
        indexes :active_for_po, type: :boolean
        indexes :upc, type: :text
        indexes :ean, type: :text
        indexes :manufacturer_id
        indexes :on_closeout, type: :boolean
        indexes :onhand, type: :integer
        indexes :published_on
        indexes :category_id, type: :text
        indexes :item_type, type: :text, analyzer: 'my_second_analyzer'
    end
end
```

Next we create the search. If you run Item.search('something') elasticsearch would search everywhere for the word 'something', but we want to give especific fields to search for, and we want to give some values that should be matched or not in the search. In the code below we create a new function to search in especific columns. We see that in the same function we call the .search function adding the query. The bool query takes a more-matches-is-better approach, so the score from each matching must or should clause will be added together to provide the final _score for each document. Then we have the multi_match query. We search for the query in the fields 'id', 'upc', and 'ean'.

The way the multi_match query is executed internally depends on the type parameter, which can be set to:

1. best_fields: Finds documents which match any field, but uses the _score from the best field.
2. most_field: Finds documents which match any field and combines the _score from each field.
3. cross_fields: Treats fields with the same analyzer as though they were one big field. Looks for each word in any field.
4. phrase: Runs a match_phrase query on each field and uses the _score from the best field.
5. phrase_prefix: Runs a match_phrase_prefix query on each field and combines the _score from each field.

With the match: query we check the value of the column, and with the must_not query we check the opposite. In the must_not query we have the range: query. Matches documents with fields that have terms within a certain range.

1. gte => Greater-than or equal to
2. gt => Greater-than
3. lte => Less-than or equal to
4. lt => Less-than

```
def self.search_single_item(query)
    self.search({
    query: {
        bool: {
            must: [
            {multi_match: {
```

```
                query: query,
                type: "phrase",
                fields: [ "id", "upc", "ean"]
            }},
            {
                match: {
                active_for_po: "true"
                }
            },
            {
                match: {
                active_for_ar: "true"
                }
            },
            {
                match: {
                published_for_web: "true"
                }
            }
            ],
            must_not: [
            {
                range: {
                onhand: {
                    lt: 1
                }
                }
            }
            ]
        }
        }
    })
end
```

We see later where we are calling this search method.

Same as the first search method we add the second search method. But this time we call the .search method from elasticsearch given the query, type and size parameters. Elasticsearch returns by default the first 10 documents if we do not define that. In this case we want the first 10000 documents. And we add the query type as parameter. Later we see why. We also changed the fields to search.

```
def self.custom_search(query, type)
    __elasticsearch__.search(query: multi_match_query(query, type), size: 10000)
end

def self.multi_match_query(query, type)
    {
        bool: {
            must: [
            {multi_match: {
                query: query,
                type: type,
                fields: [ "name", "manufacturer_name",
```

```
                            "group_name^20", "group_notes",
                            "category_name", "product_line_name",
                            "id", "group_id"]
            }},
            {
                match: {
                active_for_po: "true"
                }
            },
            {
                match: {
                active_for_ar: "true"
                }
            },
            {
                match: {
                published_for_web: "true"
                }
            }
            ],
            must_not: [
            {
                range: {
                onhand: {
                    lt: 1
                }
                }
            }
            ]
        }
    }
end
```

In public.html.erb we add a new search field only available for the power_user:

```
<!-- elasticsearch only for power user -->
<% if @power_user %>
    <%= form_tag(:action => 'quick_search') %>
    <table id="quickSearchTableElastic" cellpadding="0" cellspacing="0">
        <tr style="margin:0;padding:0;">
        <td class="menuInputNote">
            New search only BTI users:
        </td>
        </tr>
        <tr>
            <td>
            <%= hidden_field('quick_search', 'type', :value => 'elastic') %>
            <%= text_field('quick_search', 'query',
                        {:class => 'quickSearch',
                        :accesskey => "F"}) %>
            </td>
            <td>
                <%= submit_tag "GO", :class => 'go' -%>
            </td>
        </tr>
```

```
            <tr style="margin:0;padding:0;">
            <td colspan="2" class="menuInputNote">
                by BTI Item #, Description, Vendor Part #,  UPC or EAN
            </td>
        </tr>
    </table>
    </form>
<% end %>
```

It's the same code as the code above for the search field, we changed the id and the value of the hidden field. We need the same design, so in home.css, bti.css, and print.css:

```
#quickSearchTableElastic {
    margin:0 12px;
}
```

In public.html.erb we changed the type of the hidden field, now we have to add the if statement for this type. In public_controller.rb in the quick_search method we add the if statement below:

```
elsif @search_type == "elastic"
    redirect_to elastic_search_url(:query => URI.encode(routes_encode(@query)))
```

and in routes.rb to match the redirect:

```
match 'public/elastic/(/:query(/:show_group))' => 'public#list',
    :as => :elastic_search,
    :defaults => {
        :query        => 'nothing',
        #:show_group   =>  '',
        :search_type  => 'elastic'
}
```

In public_controller.rb in the list method below the the elsif == 'combined', we add the code below for the elsif == 'elastic':

```
# this part is for the elastic search
elsif @search_type == "elastic"
    query = routes_decode(params[:query])
    query = query.sub('-', '')
    single_item = Item.search_single_item(query).results

    if (single_item.count.to_i == 1)
        hash = {:action => :item,
                :id => single_item[0].id,
                :track => 'true'}
        redirect_to hash
        return
    end
```

```
    if session[:list_groups] == nil
        temp_crit = {}
        temp_crit[:partno] = query.clone
        partno_groups = Group.ids_for_search(temp_crit)

        @criteria[:itemno] = fmt_itemno_query(query.clone)
        if partno_groups.size > 0
            @criteria[:partno] = query.clone
        end
        @criteria[:descript] = fmt_descript_query(query.clone)
        @criteria[:elastic] = true
    else
        @criteria = session[:list_criteria]
    end
@title = combined_search_title(query.clone)
```

First we search with elasticsearch for one Item. If the result is one to get the id we need to add the [0] to the single_item result. Elasticsearch returns an array with all results. What is left is the same process as above, but we change the @criteria[:combined_search] to @criteria[:elastic].

In the same function we change the line where we are searching for all group id's. If we are searching with elastic in group.rb the result will be different, and we have to collect with the group_id instead of the id.

```
if @criteria[:elastic]
    @group_ids = Group.ids_for_search(@criteria).collect{|g| g.group_id}
else
    @group_ids = Group.ids_for_search(@criteria).collect{|g| g.id}
end
```

in the same function we add code below for the paging:

```
elsif @search_type == 'elastic'
      @paging_url = elastic_search_url(:only_path => true,
                :query => URI.encode(routes_encode(params[:query])),
                :show_group => 'nogroup')
```

In group.rb we have to add code below to search with elastic for all the group id's in the ids_for_search function:

```
if hash[:elastic]
    groud_ids = Item.custom_search(hash[:itemno], "phrase_prefix").to_a.uniq!{ |i|
i.group_id }
    if groud_ids.count > 1
        groups = groud_ids
    else
        groups = Item.custom_search(hash[:itemno], "best_fields").to_a.uniq!{ |i|
            i.group_id }
    end
else
```

```
    groups = self.find_by_sql([query] + parameters + ft_parameters)
end
```

Here we see that we add the type "phrase_prefix" as a parameter. The answer of elasticsearch is a object as we know. with .to_a we add all results to an array. This array contains all Items with the given query. Then we clean the array from duplicates. Now we have an array with Items objects. If we get no results, we change the type parameter to best_field and search again.

In the functions: items_for_search, groups_for_list, and for_list, where we do if hash[:partno], if hash[:descript], and if hash[:itemno] we add:

```
&& !hash[:elastic]
```

With this statement we find all items with sql as we find it with the elasticsearch.

Now we add the rake task to create the index. In /lib/tasks we add the elastic_index.rake file with the code below:

```
namespace :elastic_index do
  task delete_item_index: :environment do
    puts "Delete item index"
    Item.__elasticsearch__.delete_index!
    puts "Index deleted"
  end
  task create_item_index: :environment do
    Item.__elasticsearch__.create_index! force: true
    puts "Create and import index for Item"
    puts "Please be patient it takes a few minutes"
    Item.import
    puts "Finished"
  end
end
```

In create_item_index first we create the index and then with Item.import we import all items and create the documents. This function we call only ones after deplyoing the branch. It will take some minutes to import all items.To call the rake task just type code below in the commmand line:

```
rake elastic_index:create_item_index
```

and the command below we only use to delete the index (documents) before we run the create if we must.

```
rake elastic_index:delete_item_index
```

Next in the same folder we create the elastic_update_check.rake file with the code below:

```
namespace :elastic_update_check do
  desc 'check table inventory if there are any items updated or created'
  task update_check: :environment do
    time = DateTime.now
    items_to_update = Item.where("updated_at > ?", time-10.minutes)
    items_to_index = Item.where("created_on > ?", time-10.minutes)

    items_to_update.each do |item|
      item.__elasticsearch__.update_document
    end

    items_to_index.each do |item|
      item.__elasticsearch__.index_document
    end
  end
end
```

If we update a item in the rails app we have the callback function added in the Item class and elasticsearch takes care of the reindexing or create a new index for a new item. But if we update a row in the inventory database, elasticsearch does not know that. So we check the items and if we had a change we see that in the updated_at column. If there was a update or a new created item (we see that in the created_on column) and we run the update document or the index document command for elasticsearch. We assume that this rake runs every 10 minutes. To run the rake task:

```
rake elastic_update_check:update_check
```