ELASTICSEARCH SEARCH SYNTAX AND BOOLEAN AND AGGREGATION SEARCHES

Here we explain how to do searches in ElasticSearch (ES). ES has a seemingly endless list of search options, which can seem overwhelming. So we will start with some simple examples and build from there.

We look at these items:

Prerequisites

If you have not installed ElasticSearch, you can follow these instructions to do that. And if you could also review this cheat sheet which provides some introductory.

Not that we are not using Kibana in these examples. Kibana is an optional graphical front-end for ElasticSearch. How to use KIbana is a topic unto itself.

Data

We need some data. So:

1. Download this program to create student data. Change the URL in the program to point to your instance of ElasticSearch: http://**(your IP)**:9200/universities/universities/.
2. Then download this list of universities. Select the **csv** option and unzip the downloaded file. Copy the file **InstitutionCampus.csv**. Then download this program to load that data.

Run both programs. Both will run for several minutes since we want to create a large amount of data.

These programs create two different types of data, **students** and **universities**, in the same index, **universities**. We do this because with join type nested queries the data must be in the same index.

Basic Search Syntax

In order to get started, you need to understand something about the ES search syntax, aka Lucene Query. Here is a list of options:

search type	example
free text, meaning no field specified	"tober" matches "October" found anywhere in the document. Usually this type of parameter-less query is written into the Kibana screen (i.e., the graphical front-end to ElasticSearch) or as a curl parameter, as in: curl -XGET --header 'Cosx:9200/universities/_search?q=DON
name of field	"school":"Harvard"
range	"" for numeric fields
boolean: and, or	"tall or small". You can also leave off the word and write: "(tall small)"
scope multiple fields	"school.*":"(North South)" would query **school.name** and **school.location**.
test for value	"exists" : { "field" : "grades" } tests whether there is a field called **grades**.
wild cards: ? for one character and * for one or more	"Sou?h" matches "South". "So*h" matches "South" as well.

Boolean Searches

With all the available options, there are multiple ways to achieve the same results, like finding schools in North Carolina.

This search finds all universities with both **North** and **Carolina** in the **ParentName** field.

```
curl -XGET --header 'Content-Type: application/json'
http://parisx:9200/universities/_search/?pretty=true -d '{
      "query" : {
    "bool": {
   "must":
   }
  }
}'
```

results in:

```
      {
        "_index" : "universities",
        "_type" : "universities",
        "_id" : "fc7549cbac32fc5770c9cb7bdc72d30de9fdc5a2",
        "_score" : 10.899638,
        "_source" : {
          "ParentName" : "North Carolina Wesleyan College",
          "DapipId" : "133942003",
          "OpeId" : "",
          "LocationType" : "Additional Location",
          "AdminPhone" : "",
          "Fax" : "",
          "LocationName" : "Raleigh",
          "Address" : "2000 Perimeter Park Dr, Morrisville, NC 27560",
          "AdminName" : "",
          "ParentDapipId" : "133942",
          "UpdateDate" : "",
          "AdminEmail" : "",
          "GeneralPhone" : ""
        }
      },
```

You can limit the output to just certain fields using the **_source** parameter as shown below.

```
curl -XGET --header 'Content-Type: application/json'
http://parisx:9200/universities/_search/?pretty=true -d '{
      "_source" : ,
      "query" : {
    "bool": {
   "must":
   }
  }
}'
```

Results in:

```
{
        "_index" : "universities",
        "_type" : "universities",
        "_id" : "455f7ec172260cae0afad78c6b532920d65876d3",
        "_score" : 5.5798583,
        "_source" : {
          "ParentName" : "Florida State University"
        }
      }
```

Aggregation

Aggregation lets you group data to do counts.

In the program massAdd.py, change this index in this line of code:

```
url="http://parisx:9200/universities/universities/"
```

to a different index:

```
url="http://parisx:9200/students/students/"
```

And run that program to load data to the student index.

Then run this search:

```
curl -XGET --header 'Content-Type: application/json'
http://parisx:9200/students/_search/?pretty=true -d '{
  "size": 0,
  "aggs": {
    "group_by_school": {
      "terms": {
        "field": "school.keyword"
      }    }
  }
}'
```

Results in:

```
"aggregations" : {
    "group_by_school" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" :
    }
```

In SQL, this is the same as:

```
select *, count(*) from schools group by schools
```

This shows you how many students are at each school.

Notice below that we have to add the word **keyword** to **school.keyword**. If we did not use the **keyword** in **school.keyword** it would produce this error:

> Fielddata is disabled on text fields by default. Set fielddata=true on in order to load fielddata in memory by uninverting the inverted index. Note that this can however use significant memory. Alternatively use a keyword field instead.

Basically ElasticSearch is saying that doing aggregation on the text fields would require calculating extra data and holding that in memory. That's not needed for ordinary search queries.