1 ElasticSearch query language

ElasticSearch has its own query language, more documentation can be found here

1.1 Anatomy of a query

In most cases, an ElasticSearch query is composed of :

- a **header** defining :
  - which ***method*** is to be used : `GET`, `POST`, etc.
  - which ***index*** (i.e. which entity of the database written in snake_case) is going to be queried. Not necessary if all indexes will be queried.
  - what ***type*** of search is to be made (`_search` most ofthe time)
- a **body** defining (among other things) :
  - the ***fields*** that are going to appear in the results (`_source`)
  - the ***filters*** that are going to narrow the number of results matching those filters (`query`)
  - different properties of the results (size of the results, index from which to begin, etc.)

Tips & tricks The body of a query needs to be a correctly formatted JSON string, even using single quotes instead of double is considered to be an error. The `dev tools` tab in Kibana interface offers help for automatic indentation and autocompletion features that can be very handy.

1.2 Get all records

To get all the records of the entire database :

```
GET _search
```

To retrieve all records from an index (primary source in this case) :

```
GET primary_source/_search
```

Which is equivalent to :

```
GET primary_source/_search
{
    "query": {
        "match_all": {}
    }
}
```

1.3 Simple matching query

Exact term match

All the works that contain the string “tabule” in their title.

Note that the case of the letters doesn’t matter, it will match “Tabule” as well. In addition to that, in this configuration, a sub-string will not match the same as the entire string (“tabu” will not match “tabule”) ; the string is treated as a complete word.

```
GET work/_search
{
    "query":{
        "match": {
            "title": "tabule"
        }
    }
}
```

All the original items that are associated with a primary source that is kept in the library that have `2` as id.

```
GET original_text/_search
{
    "query": {
        "match": {
            "primary_source.library.id": "2"
        }
    }
}
```

Multiple terms match

All original items that have in their title either the word “solis”, either the word “lune”.

In this configuration, each string separated by a space is treated individually and the operator used to connect them is `OR`. In other words, the more you put terms, the more you will match original items.

```
GET original_text/_search
{
    "query":{
        "match": {
            "original_text_title": "solis lune"
        }
    }
}
```

To get a response where each terms given are independently going to filter the result (same kind of behavior as a Google query), you need to specifies the operator to be `AND`.

```
GET original_text/_search
{
    "query": {
        "match": {
            "original_text_title": {
                "query": "lune solis",
                "operator": "AND"
            }
        }
    }
}
```

1.4 Adding some margin of error

Fuzziness The fuzziness allows a certain amount of inaccuracy to be accepted.

> All library that approximately have the string “natonale” in their name.

```
{
    "query": {
        "match": {
            "library_name": {
                "query": "natonale",
                "fuzziness": "auto"
            }
        }
    }
}
```

You can set the fuzziness to `1` or more but the `auto` settings allows a number of letters that do not match, proportional to the length of the term to be searched.

1.4.1 Full text search on an index

To allow search on every field of an entity, the query has to be set to `multi_match`.

> All edited texts that have approximately the string “lune” in one of them fields.

```
GET edited_text/_search
{
    "query": {
        "multi_match": {
            "query": "lune",
            "fuzziness": "auto"
        }
    }
}
```

As is, those kind of requests are deprecated because no fields are specified : ElasticSearch encourages to list the fields you want the query to be executed on. The `fields` allows to reduce noise in the results and to take less time.

> All primary sources that match approximately the strings “vatican” and “latin” in the list of fields specified.

```
GET primary_source/_search
{
    "query": {
        "multi_match": {
            "query": "vatican latin",
            "fuzziness": "auto",
            "operator": "and",
            "fields": [
                "shelfmark",
                "digital_identifier",
                "kibana_name",
                "tpq.keywork",
                "taq.keywork",
                "prim_type",
                "library.kibana_name",
                "original_texts.kibana_name",
                "original_texts.table_type.kibana_name",
                "original_texts.place.kibana_name",
                "original_texts.historical_actor.kibana_name",
                "original_texts.script.script_name",
```

```
                "original_texts.language.language_name"
            ]
        }
    }
}
```

Notice that on the fields `tpq` and `taq` that are typed as integer, a string query cannot be performed. In order to query those fields as well, you must add `.keyword` after the field name : it corresponds to the field but typed as a string.

Wildcards To find some more documentation for wildcard queries, click here.

1.5 Defining the source

If you are not interested in all the metadata (i.e. the content of the fields) associated with the entity you want to query, it is possible to set a list of fields that are going to appear in the response.

> Only the shelfmark, and the library name of all primary sources that are a manuscript

```
GET primary_source/_search
{
    "_source": [
        "shelfmark",
        "library.library_name"
    ],
    "query": {
        "match": {
            "prim_type": "ms"
        }
    }
}
```

Note that if a record in the result do not have some information you asked for (let's say, the primary source isn't associated with a library, thus doesn't have a `library.libray_name`), the result object will not have the key for this precise field. Instead of looking like that :

```
"_source" : {
    "library" : {
        "library_name" : "Vatican Library"
    },
    "shelfmark" : "Vat. Pal. Lat. 1376"
```

```
}
```

It will look like :

```
"_source" : {
    "shelfmark" : "Vat. Pal. Lat. 1376"
}
```

1.6 Special queries

1.6.1 Range queries

Range queries can be made on fields that are numbers (even if the field is typed as a string but contains integer) in the Kibana interface, but does seem to only work on integer/float/date typed field when using ajax.

> All the primary source that have an edition date between 1400 and 1500

```
GET primary_source/_search
{
    "query": {
        "range": {
            "date": {
                "gte": 1400, // greater than
                "lte": 1500 // less than
            }
        }
    }
}
```

If you want to make a range query on a date typed field (fields that end with `_date`, you can use some ElasticSearch tools for date math (those as well, seems to cause problem when used with ajax) :

> All original items that have been created between 1000 years before today and 25 years after 1500

```
GET original_text/_search
{
    "query": {
        "bool": {
            "must": [
                {
```

```
                    "range": {
                        "tpq_date": {
                            "gte":"now-1000y"
                        }
                    }
                },
                {
                    "range": {
                        "taq_date": {
                            "lte": "1500-01-01||+25y"
                        }
                    }
                }
            ]
        }
    }
}
```

1.6.2 Geo-distance queries

The fields named `location` holds information that is treated as geo point by ElasticSearch : `geo_distance` queries can be executed on them.

> All works that have been conceived around 100km from 48 of latitude and 2 of longitude. **NB** : in this syntax, `longitude` comes before `latitude`.

```
GET work/_search
{
    "query": {
        "geo_distance": {
            "distance": "100km",
            "place.location": [2,48]
        }
    }
}
```

The same query can be formulated more explicitly with this syntax :

```
GET work/_search
{
    "query": {
        "geo_distance": {
            "distance": "100km",
            "place.location": {
                "lat" : 48,
                "lon" : 2
            }
        }
    }
}
```

1.7 Combining multiple clauses

Every filter you want to combine to build a query can be add with this kind of structure :

```
{
    "query": {
        "bool": {
            "must": [
                {
                    // filter 1
                },
                {
                    // filter 2
                }
            ]
        }
    }
}
```

Putting all together

The shelfmarks of all early printed primary sources that contains an original item that were created near Paris (lat : 48, long : 2).

```
GET original_text/_search
{
    "_source": [
        "primary_source.shelfmark"
    ],
    "query": {
        "bool": {
            "must": [
                {
                    "geo_distance": {
                        "distance": "100km",
                        "place.location": [2,48]
                    }
                },
                {
                    "match": {
                        "primary_source.prim_type": "ep"
                    }
                }
            ]
        }
    }
}
```