Recherche de documents

Rechercher avec l'API Rest 1/2

- L'API Rest « Search » permet de rechercher des documents
  - dans un, plusieurs ou tous les index (**multiindex**)
  - sur un, plusieurs ou tous les types de documents (**multitypes**)
  - avec une ou plusieurs requêtes de recherche (**multisearch**)
- Utilisation

```
POST  /index(s)/type(s)/_search
{"query"  :  {...}}
```

  - Le contenu de la requête est écrit en **JSON**
  - Le champ `query` contient la requête de recherche
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html

Rechercher avec l'API Rest 2/2

- Il existe aussi un raccourci pour les recherches textuelles

```
GET  /index(s)/type(s)/_search?q=titre:Misérables
```

- Plus simple, pratique pour tester rapidement une recherche
- Accepte de nombreux paramètres
  - `q` : requête de recherche (de type `query_string`)
  - `df` : champ par défaut pour la requête
  - `explain` : affiche les informations de calcul de la pertinence
  - Ainsi que les champs `from,size,fields,sort`

Recherche multi-index

- Recherche sur un index et un type de document

```
POST  /formation/book/_search
{  "query"  :  {
        "match_all"  :  {}}}
```

- Recherche sur plusieurs index et un type de document

```
POST  /formation1,formation2/book/_search
{  "query"  :  {
        "match_all"  :  {}}}
```

- Recherche sur tous les index et un type de document

```
POST  /_all/book/_search
{  "query"  :  {
        "match_all"  :  {}}}
```

Recherche multi-index avec wildcards

- Recherche sur les index commençant par « formation »

```
POST  /formation*/_search
{"query"  :  {"match_all"  :  {}}}
```

- Possibilité d'inclure/exclure des patterns de noms d'index :

```
POST  /+test*,-test1/_search
{"query"  :  {"match_all"  :  {}}}
```

- Il est aussi possible de contrôler le comportement de l'expansion :
  - `ignore_unavailable`=true, false
  - `allow_no_indices`=true, false

Recherche multi-type

- Recherche sur un index et plusieurs types de documents

```
POST  /formation/book,dvd/_search
{  "query"  :  {
        "match_all"  :  {}}}
```

- Recherche sur plusieurs index et plusieurs types de documents

```
POST  /formation1,formation2/book,dvd/_search
{  "query"  :  {
        "match_all"  :  {}}}
```

- Recherche sur tous les index et tous les types de documents

```
POST  /_all/_search
{  "query"  :  {
        "match_all"  :  {}}}
```

Résultats de recherche

- La réponse à une requête de recherche est au format JSON


```
{ "_shards": {"failed": 0, "successful": 5, "total": 5},
  "hits": {
    "hits": [
      {"_index": "formation", "_type": "book", "_id": "sIMPUEKOSNqmus6pnXzUag",
          "_score": 1.0,
          "_source": {
            "content": "Les Misérables...",
            "lang": "fr",
            "title": "Les Misérables"},...},...],
    "max_score": 1.0, "total": 3},
  "timed_out": false, "took": 1}
```


  - Nombre de shards interrogés
  - Nombre total de résultats
  - Temps d'exécution de la requête en ms
  - Tableau des résultats
  - Pertinence du résultat

Pagination

- Il est possible de contrôler le nombre de résultats récupérés

```
POST  /index(s)/type(s)/_search?from=0&size=100
{"query"  :  {…}}
```

  - `from` : position du premier résultat à retourner (défaut 0)
  - `size` : nombre de résultats à retourner (défaut 10)
  - from + size < 10000
- Pour un très grand nombre de résultats, préférer le **scroll**
  - Fonctionnement similaire à un curseur de base de données
  - Paramètre de requête `scroll` = temps de conservation des résultats
  - La réponse contient les premiers résultats et un `scroll_id`
  - Pour obtenir les résultats suivants, requêter avec le `scroll_id` reçu

Pagination avec Scroll

- Le scroll permet de conserver un résultat un temps donné, l'état de l'index est figé pour cette requête
- Permet de revenir sur un résultat de recherche sans la relancer
- Le résultat est stocké en mémoire dans le cluster
- N'est pas adapté aux recherches temps réel

```
POST  /formation/book/_search?scroll=10m
{  "size"  :  10,
    "query":  {"query_string":  {"query":  "title:hunger"}}}
```

Réponse

```
{  "_scroll_id":  "cXVlcnlUaGVuRmV0Y2g7NTsxMDE4MDo0...",
    "hits":  {  ...  }
}
```

Appel suivant

```
GET  _search/scroll?scroll=10m&scroll_id=cXVlcnlUaGVuRmV0Y2g7NTsxMDE4MDo0...
```

Structure des résultats

- Il est possible de préciser les champs des documents à retourner en résultat

```
POST  /formation/book/_search
{  "query"  :  {"match_all"  :  {}},
    "fields"  :  ["title",  "lang"]}
```

- La valeur des champs est extraite :
  - de `_source`, si la source est activée pour le document
  - du document, si la source est désactivée et le champ stocké (`store=true`)

En fonction du mapping, préciser des fields peut diminuer les performances de la recherche (accès disques multiples)

Tri des résultats 1/2

- Par défaut, les résultats sont triés par pertinence décroissante
- Il est possible de trier sur un ou plusieurs champs

```
POST  /formation/book/_search
{  "query"  :  {"match_all"  :  {}},
    "sort"  :  [
       {  "title"  :  "desc"  },
       {  "publish_date"  :  {"order"  :  "asc",  "missing"  :  "_last"}  },
       {  "category"  :  {"unmapped_type"  :  "string"}  },
       "_score"]}
```

- L'option `missing` indique ce qu'il faut faire des documents qui n'ont pas le champ souhaité. Les faire apparaître en `_first` ou en `_last`
- L'option `unmapped_type` permet de ne pas avoir d'erreur lors de la recherche d'un champ **non mappé dans un index** et donne le **type** à utiliser pour retourner une valeur de tri **sans effet**

Tri des résultats 2/2

Trier sur un champ peut consommer ***beaucoup*** de ***mémoire*** car l'ensemble des valeurs du champ sur les documents remontés sont chargés en mémoire

- Ne pas trier sur des champs string ***analysés***

Attention, si la pertinence n'est pas utilisée pour le tri (`_score`), elle ne sera pas remontée en résultat

- Il faudra utiliser l'option `"track_scores": true`

```
...
"sort":  [{"lang":  "asc"}],
"track_scores":  true
...
```

Comptage des résultats

- L'API Rest `_count` fournit un moyen simple de compter les résultats d'une recherche sans récupérer les documents
- Sans requête :

```
GET  /formation/book/_count
GET  /formation1,formation2/_count
GET  /_count
```

- Avec requête de recherche :

```
GET  /formation/book/_count?q=spring
GET  /formation/book/_count
{"query"  :  {"match"  :  {  "author"  :  "Victor  Hugo"  }}}
```

Les Queries

Langage de recherche

- Elasticsearch a un langage de définition de requêtes : **Query DSL**
  - Ce langage est écrit en **JSON**
  - Il est passé dans le champ `query` de la requête
- Structure générale :
  - Le langage définit des opérations de recherche
  - Chaque opération a ses propres paramètres
  - Certaines opérations permettent d'en combiner d'autres

```
{  "query":  {
       "search_operation_x":  {"search_parameter_x":  "value"...},
       "search_operation_y":  {"search_parameter_x":  "value"...}...}}
```

Types de requêtes

https://www.elastic.co/guide/en/elasticsearch/reference/current/querydsl.html

- `match_all`
- Full text: `match`, `match_phrase`, `multi_match`, `query_string`...
- Termes: `term`, `range`, `exists`, `missing`, `prefix`, `wildcard`, `regexp`, `fuzzy`...
- Composées: `bool`, `and`, `or`,`not`...
- "Jointures": `nested`, `has_child`, `has_parent`
- Spatiales: `geo_distance`, `geo_bounding_box`...
- Proximité, phrase: `span_near`, `span_term`...
- Scoring: `constant_score`, `function_score`, `boosting`...

Query | match_all

- Recherche qui renvoie tous les résultats

```
{  "query":  {
        "match_all":  {}}}
```

Query | query_string

- Recherche fulltext Lucene

```
{  "query":  {
       "query_string":  {
          "fields":  ["content","title^2","author.*"],
          "query":  "Victor  AND  Hugo  OR  1862"}}}
```

- Syntaxe très ***stricte***
- Options
  - La recherche est effectuée par défaut sur le champ `_all`
  - Possibilité de préciser les champs sur lesquels rechercher avec `fields`
  - Le paramètre `query` est automatiquement parsé pour construire une requête de recherche
  - Les analyseurs sont détectés et appliqués aux mots de la recherche

Query | simple_query_string

- Similaire à query_string

```
{  "query":  {
       "simple_query_string":  {
          "fields":  ["content","title^2","author.*"],
          "query":  "Victor  AND  Hugo  OR  1862"}}}
```

- Plus laxiste sur la syntaxe, ***ignore*** simplement les tokens ***invalides***

Query | match

- Recherche fulltext

```
{  "query":  {
        "match":  {"author":  "Victor  Hugo"}}}
```

- Les analyseurs sont **appliqués**
- Les motclés de la recherche ne sont **pas parsés**
  - Potentiellement moins d'erreurs
- A privilégier par rapport à `query_string`

Query | match

- Par défaut, un document `match` si au moins un des termes est présent
- Modifiable avec le paramètre `operator` à `and` ou bien à `or`
- Syntaxe complète :

```
{
   "query":  {
         "match":  {
               "author":  {
                     "query":  "Victor  Hugo",
                     "operator":  "and"
               }
         }
   }
}
```

Query | match_phrase

- Variante de la requête `match`

```
{
   "query":  {
         "match_phrase":  {
               "author":  "Victor  Hugo"
         }
   }
}
```

- La **position respective** des termes doit être **respectée**
- Introduction d'une part de souplesse possible avec le paramètre `slop` (`0` par défaut):
  - Indique le **nombre de déplacements** de termes **autorisés**

Query | match_phrase_prefix

- Variante de la requête `match` proche de la requête `match_phrase`

```
{
   "query":  {
         "match_phrase_prefix":  {
               "author":  "Victor  Hu"
         }
   }
}
```

- Le dernier terme sert de prefixe utilisé pour rechercher parmi les termes
- Peut servir à faire de l'autocomplétion "quickanddirty"
  - Avantage : rien à préparer en amont (pas de réindexation)
  - Inconvénient : solution la moins performante

Query | multi_match

Recherche fulltext sur plusieurs champs

```
{  "multi_match"  :  {
        "query"  :  "Victor  Hugo",
        "fields"  :  [  "author",  "description"  ]}}
```

- Permet de réaliser une requête `match` sur plusieurs champs en évitant une requête booléenne
- Utilise une requête `dis_max` par défaut
  - Retourne les documents qui matchent au moins une des sousrequêtes
  - Le score est le score maximum des sous requêtes
- Plusieurs stratégies possibles :
  - ***best_fields*** (défaut)
  - ***most_fields***
  - ***cross_fields***

Query | term

- Recherche les documents contenant un terme précis

```
{  "query":  {
       "term":  {"lang":  "fr"}}}
```

- Attention, **aucun analyseur** n'est **appliqué**
- Options
  - Possibilité de **booster** un résultat par rapport à un autre
  - Plusieurs requêtes `term` peuvent être combinées dans une requête `terms`

Query | prefix

- Recherche les documents commençant par un terme précis

```
{  "query":  {
        "prefix":  {
            "content":  "rom"}}}
```

- Attention, **aucun analyseur** n'est **appliqué**

Query | wildcard

- Recherche les documents contenant une expression

```
{  "query":  {
        "wildcard":  {
            "title":  "du?l*"}}}
```

- Caractère étoile * pour matcher zéro ou plusieurs caractères
- Point d'interrogation ? pour matcher un seul caractère
- Attention
  - **Aucun analyseur** n'est **appliqué**
  - **Ralentit** énormément les requêtes de recherche
  - **Ne jamais** commencer l'expression par * ou ?

Query | range 1/2

- Recherche les documents compris entre 2 bornes

```
{  "query"  :  {
       "range"  :  {
          "price"  :  {
             "gte"  :  50,
             "lt"  :  200}}}}
```

- S'applique aux dates, nombres et chaînes de caractères
- Possibilité d'inclure ou non les bornes supérieures/inférieures suivant les paramètres utilisés :
  - gt (>) / gte (≥)
  - lt (<) / lte (≤)
- Exemple ci dessus : prix compris entre 50 (inclus) et 200 (exclu)

Query | range 2/2

- `range` s'applique aussi aux dates

```
{  "query":  {
       "range":  {
          "created":  {
             "gte"  :  "2015-12-15"  }}}}

{  "query":  {
       "range":  {
          "created":  {
             "gte"  :  "now-1d/d",
             "lt"  :   "now/d"  }}}}
```

  - Les unités supportées sont: `y` (year), `M` (month), `w` (week), `d` (day), `h` (hour), `m` (minute), and `s` (second).
  - `+1d` et `-1d` signifient respectivement plus et moins une unité de type jour.
  - `/d` signifie arrondi à l'inférieur au jour le plus proche

Recherche floue 1/2

- Recherche par **similarité**
- Calculée suivant la ***distance de DamerauLevenshtein***
  - Nombre minimal d'opérations pour passer d'un mot **M1** à un autre **M2** :
    - substitution d'un caractère de **M1** en un caractère de **M2** (au même rang)
    - insertion dans **M1** d'un caractère de **M2**
    - suppression d'un caractère de **M1**
    - permutation de deux caractères **M1** et **M2** adjacents
- Exemple : distance de Levenhstein entre `avicto` et `victor`

```
`avicto` => `victo`: suppression du 1er caractère

`victo` => `victor`: insertion du 6e caractère
```

  - résultat: distance égale à 2

Recherche floue 2/2

- Dans ElasticSearch : paramètre `fuzziness` utilisable dans plusieurs requêtes (`match`,`multi_match`,`fuzzy` ...)
- Valeurs autorisées :
  - distance en valeur absolue entre 0 et 2
  - `AUTO` (valeur par défaut)
    - longueur < 3 : distance = 0
    - longueur entre 3 et 5 : distance = 1
    - longueur > 5 : distance = 2

La distance maximum autorisée est **2**, les valeurs supérieures ne sont pas prises en compte

Query | fuzzy 1/2

- Equivalent de la requête `term` avec flou

```
{
   "query":  {
         "fuzzy":  {
               "author":  {
                     "value":  "vitcor",
                     "fuzziness":  2,
                     "prefix_length":  1
               }
         }
   }
}
```

- **Aucun analyseur** n'est **appliqué**
- Paramètres impactant les performances :
  - `prefix_length`: nombre de caractères non modifiés en début de terme
  - `max_expansions` : nombre max de termes similaires générés
- En général, une requête `match` avec `fuzziness` est plus simple

Query | fuzzy 2/2

- Peut être utilisée avec des nombres ou dates (équivalent à une Range Query)

```
{ "query": {
        "fuzzy" : {
            "price" : {
                "value" : 100,
                "fuzziness" : 5 }}}}
```

- ou bien

```
{ "query": {
        "fuzzy" : {
            "created" : {
                "value" : "2010-02-05T12:05:07",
                "fuzziness" : "1d" }}}}
```

- Dates : unités de temps autorisées => `d`, `h`, `m`, `s`

Query | bool 1/2

- La requête de type `bool` permet de combiner plusieurs requêtes
- Les requêtes sont combinées à l'aide de 3 paramètres
  - `must` : le document doit correspondre au(x) clause(s) de la requête
  - `must_not` : le document ne doit pas correspondre au(x) clause(s) de la requête
  - `should` : le document peut correspondre au(x) clause(s) de la requête
- Si aucune clause `must` n'est déclarée, le document doit correspondre à au moins 1 clause `should`
  - Paramètre `minimum_number_should_match` définit le nombre de clauses à respecter pour que le document soit retenu comme résultat

Query | bool 2/2

- Exemple de requête booléenne

```
{ "query": {
    "bool": {
      "must": [{
        "match": {
          "titre": "spring in action" }},{
        "term": { "tag": "spring" }}],
      "must_not": {
        "range": {
          "annee": { "lte": 2010 }}},
      "should": [{
        "term": { "tag": "java" }},{
        "term": { "categories": "developpement" }}]}}
```

Query | exists et missing

- `exists` permet de filtrer les résultats en fonction de l'existence d'un champ (le champ doit avoir une valeur)

```
{  "query":  {
        "exists":  {"field":  "soustitres"}}}
```

- `missing` permet de filtrer les résultats en fonction de l'absence d'un champ (le champ n'existe pas ou n'a pas de valeur)

```
{  "query":  {
        "missing":  {"field":  "soustitres"}}}
```

Query | and, or et not

- `and` et `or` permettent de combiner plusieurs filtres à appliquer aux résultats d'une recherche
- `not` permet d'exclure des documents des résultats de la recherche

Préférer `bool` pour des raisons de pérennité et de performances

Query | index, type et ids

- `type` permet de préférer certains type de document

```
{  "query":  {
        "type":  {"value":  "livre"}}}
```

- `ids` permet de sélectionner certains ids

```
{  "query":  {
      "ids"  :  {
         "type"  :  "livre",
         "values"  :  ["4",  "12",  "38"]  }}}}
```

- On peut aussi utiliser un `term` sur les pseudochamps `_index`,`_type`,`_id`

```
{  "query":  {
      "term":  {
         "_type":  "livre"  }}}
```

Query | filter 1/3

- On utilise une requête de type `bool` ou `constant_score` avec une clause `filter` (ou `must_not`)

```
{ "query": {
      "bool": {
         "must": {
            "match": {
               "resume" : "victor hugo"}},
         "filter": {
            "range": {
               "price": {"gt": 25, "lt": 50} }}}}}
```

- ou bien

```
{ "query": {
      "constant_score": {
         "filter": {
            "range": {
               "price": {"gt": 25, "lt": 50} }}}}}
```

- Préférer une requête `bool` à `filtered`

Query | filter 2/3

Les filtres sont très intéressants d'un point de vue performance

- **Pas de calcul de score**
  - Un filtre ne participe pas au calcul du score
  - Il est généralement plus rapide qu'une `query` classique
  - A préférer s'il n'y a pas de score à calculer
  - A préférer avec les opérateurs sans analyse `term`, `range`, `exists`...
  - Permet de limiter le sous ensemble de documents sur lequel sera calculé le score

Query | filter 3/3

- **Mise en cache**
  - Mise en cache automatiquement sous forme de BitSet
  - Quelques exceptions `script`, `range` + `now`

Doc Id	A	B	C	D	E	F	G	H
Term	1	0	0	1	1	0	1	0

- **Combinables**
  - Utiliser un `bool` plutôt que `and`,`or` ou `not`
  - Prioriser les filtres du moins au plus coûteux

Doc Id	A	B	C	D	E	F	G	H
Term	1	0	0	1	1	0	1	0
Range	0	1	0	1	0	1	1	0
And	0	0	0	1	0	0	1	0