# 8 - Introducing Search

Monday, 24 March, 2025 21:24 Tarde

**OVERVIEW**

- Há busca estruturada e não-estruturada
  - Estruturada -- match exato, sem score de relevância
  - Não-estruturada -- por relevância, aproximação.
- Precision: densidade de documentos relevantes
- Recall: quanto de todos que eu gostaria de retornar foram retornados.

**COMO FUNCIONA?**

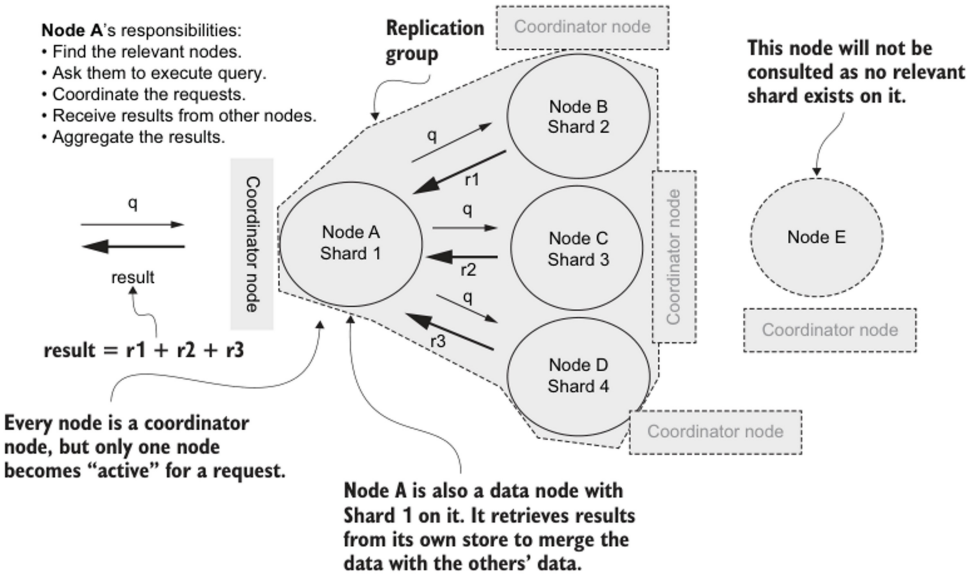


Figure 8.1 A typical search request, and the mechanics of how a search works

**MOVIE SAMPLE DATA**

- Exemplo de index de filmes

**SEARCH FUNDAMENTALS**

- _search endpoint
  - GET com parametros
  - POST com payload
    - Query DSL (Domain Specific Language)
    - Melhor pra buscas mais complexas
- Toda query ocorre em um de dois contextos:
  - *Query context*:
    - Gera score de relevância
  - Filter context:
    - Sem score, só retorna ou não documentos.
    - Poupa recursos da máquina, não requer passar por scoring, usa caching

**ANATOMIA DE UMA REQUISIÇÃO E UMA RESPOSTA**

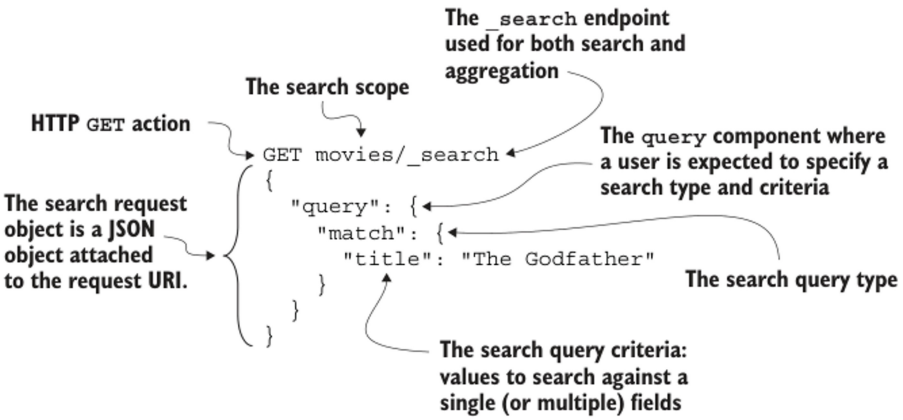


Figure 8.2 Components of a search request

- GET e POST são equivalentes na prática, mas se costuma usar POST quando há body.
- Se não fornecermos um index na busca, pressupõe-se que são todos.

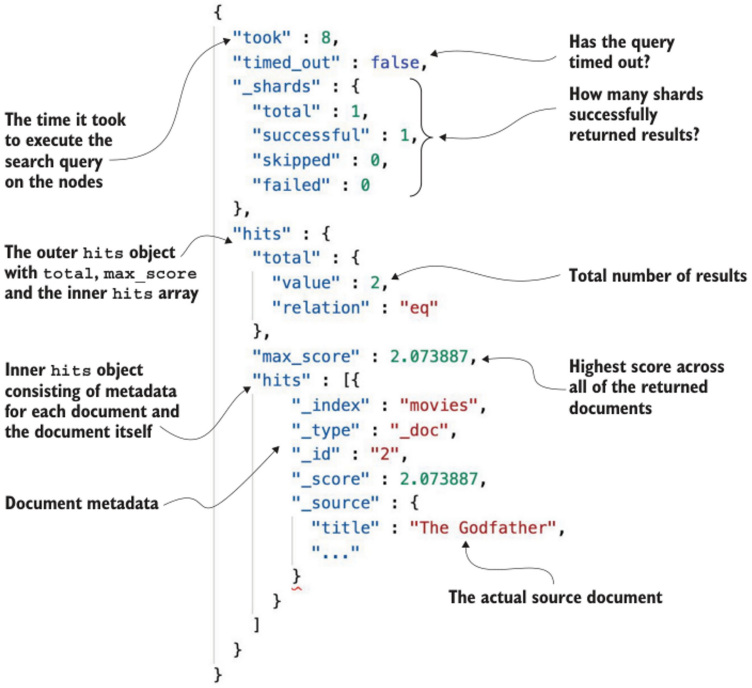


Figure 8.3 Constituents of a search response

**URI REQUEST SEARCHES**

- Busca por URI:
  - Busca com params, menos comum e menos flexível.
  - *GET movies/_search?q=title:Godfather Knight*
  - Passando o parâmetro explain=true o score é explicado.
  - Permite AND operator (e outros booleanos)
  - Busca por múltiplos campos ao mesmo tempo.
  - Podemos editar o *default_operator* para ser AND e não OR, para os vários critérios.
  - Busca por desigualdades (>= < etc)
- Podemos passar "query_string" na query por body que funciona igual ao ?q= pelo URI (não recomendado)

**QUERY DSL**

- Baseado em JSON
- Bastante flexível

- Define um "query" object:
  - Dentro dele é aplicado a lógica da busca.

**Listing 8.12 Query DSL sample query**

```
GET movies/_search
{
  "query": {
    "multi_match": {
      "query": "Lord",
      "fields": ["synopsis","title"]
    }
  }
}
```

- Dá pra usar com CURL normalmente sendo o body de um POST/GET
- Para buscas de agregados (analytics), usamos o objeto "aggs" no lugar do "query":

**Listing 8.14 Aggregation query written in Query DSL format**

```
GET movies/_search
{
  "size": 0,
  "aggs": {
    "average_movie_rating": {
      "avg": {
        "field": "rating"
      }
    }
  }
}
```

- Leaf Query:
  - Busca simples que busca um dado objetivo.
  - Um predicado só
- Query Composta:
  - Complexo, vários critérios ao mesmo tempo
  - Combina várias leaf queries

**SEARCH FEATURES**

- Podemos manipular a forma de retornar os resultados
- Paginação:
  - 10 resultados por default
  - Parâmetro size
    - Número de resultados retornados
  - Importante calibrar para não sobrecarregar client ou servidor
  - Máximo é 10_000 por default
    - Pode ser alterado nos settings do index
    - Aumentar é pouco recomendável
  - Parâmetro from
    - Só retorna a partir da n-ésima página, ignora as anteriores
  - Se houver muitos resultados, é melhor usar o parâmetro *search_after* e não *from* ou *size* por questões de desempenho.
- Highlighting:
  - Destaca fields especificados
  - Por default, envolve em tags :
    - Pode ser editado

**Listing 8.19 Highlighting results when matched**

```
GET movies/_search
{
  "_source": false,        <-- Suppresses the source to be returned
  "query": {
    "term": {
      "title": {
        "value": "godfather"
      }
    }
                           | Includes a highlight object along
```

**Listing 8.19 Highlighting results when matched**

```
GET movies/_search
{
  "_source": false,          <-- Suppresses the source to be returned
  "query": {
    "term": {
      "title": {
        "value": "godfather"
      }
    }
  },"highlight": {           <-- Includes a highlight object along with the fields to be highlighted
    "fields": {
      "title": {}            <-- The field we want to highlight
    }
  }
}
```

- Explaining:
  - Flag explain=true
    - Declarado no mesmo escopo que o objeto query
  - Explica como foi calculado o score de relevância para os resultados.
    - Boost * IDF (inverse document frequency) * TF (term frequency)
  - Também podemos usar a api *_explain*
- Sorting:
  - Declarado no mesmo escopo que o objeto query
  - Permite editar critério de ordenamento:
    - Default é score decrescente
  - Se _score não for o critério de ordenamento, não é computado.
    - Mas podemos gerar score mesmo assim com flag "*track_scores*"
  - Podemos ordenar por múltiplos campos
    - Campos posteriores servem de desempate.
- Manipulating Results:
  - Podemos omitir o conteúdo dos documentos encontrados:
    - Parâmetro _source=False
  - Podemos devolver só um subconjunto dos campos dos documentos, em vez do documento inteiro.
    - Usamos o campo "fields" para isso e um array com os nomes dos campos.
      - Permite uso de wildcard
  - Podemos manipular o resultado com um script antes de retorná-lo
    - Usamos o campo "script_fields", que inclui os fields e os scripts respectivos.
  - Podemos definir _source como uma lista de fields:
    - Incluir ou excluir fields
      - Aceita wildcard
  - Podemos priorizar os resultados de alguns índices mais do que outros:
    - Parâmetro "indices_boost"
      - Multiplica-se um coeficiente para o score dos resultados daquele index.

## Summary

- Searching can be categorized as structured and unstructured search types.
- Structured data works with non-text fields like numeric and date fields or fields that are not analyzed during indexing time and produce binary results (either they exist or they don't).
- Unstructured data deals with text fields expected to have a relevancy score. The engine scores the results based on how well the resultant documents match the criteria.
- We use term-level search for structured queries and full-text search for unstructured data.
- Every search request is processed by a coordinator node. Coordinator nodes are responsible for asking other nodes to execute the query, return partial data, aggregate it, and respond to the client with a final result.
- Elasticsearch exposes a `_search` endpoint for queries and aggregations. We can invoke the `_search` endpoint by either using a URI request with parameters or building a full request using a special syntax called Query DSL.
- Query DSL is the preferred choice for creating search queries. We can construct a plethora of queries, including advanced queries, using Query DSL.
- Query DSL allows us to create leaf and compound queries. Leaf queries are simple search queries with a single criterion. Compound queries are used for advanced queries built with conditional clauses.
- Crosscutting features are available for most types of queries: pagination, highlighting, explanation of the scoring, manipulation of the results, and so on.