4 - Mapping

Sunday, 2 February, 2025 11:29 Manhã

- Schema é inferido pelo ElasticSearch
  - Mas é melhor declarar explicitamente antes
- Mapping é o que cria a definição do schema
  - Um mapping por index
    - (Um field pode ser multi-tipo)

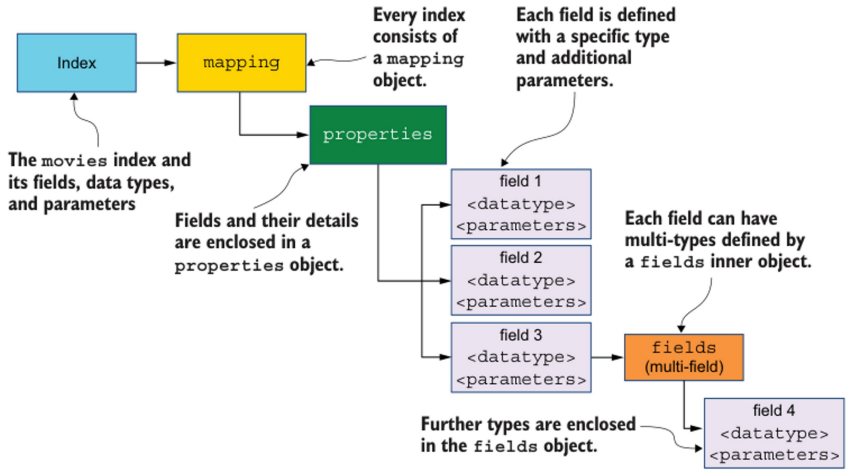

Figure 4.1 Anatomy of a mapping schema

- Etapas quando damos PUT num documento novo em index novo:
  - Um novo índex é criado com tudo default
  - Um schema é criado para esse index baseado nos data types inferidos
    - Mapping dinâmico
  - Documento é indexado e guardado
  - Documentos seguintes pulam as primeiras etapas
- Podemos dar GET no mapping em si

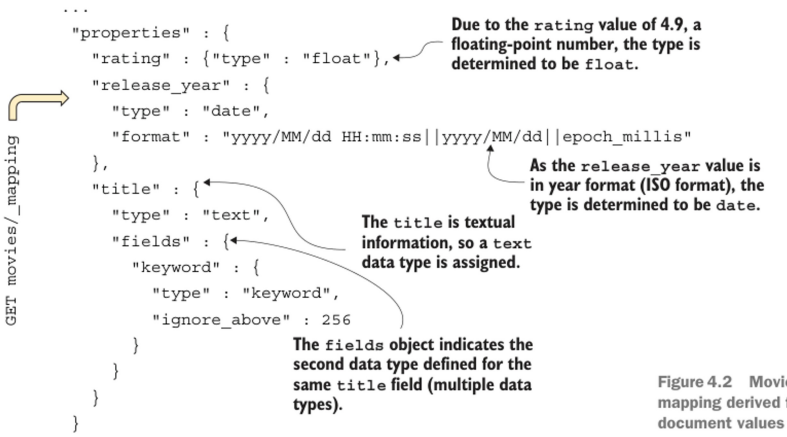

Figure 4.2 Movie index mapping derived from document values

- Fields multi-tipos
- Tipo data type **keyword** não é fatiado em tokens, é tratado como um token só.
- Mapping dinâmico pode errar, melhor só usar em testes.
  - E.g. números envoltos em aspas são tratados como strings

        - Não vai ser ordenável
- Texto pode ser ordenável se configurarmos *fielddata* como true
    - Mas *field data* é computacionalmente caro
    - Melhor usar keywords no lugar
    - Todo field pode ter um segundo tipo keyword
        - Daí ordenamos os ***field.keywords***
- Única data padrão reconhecível automaticamente é YYYY-MM-DD

- **<u>Mapping explítico</u>**
    - Geralmente recomendado
- Podemos atualizar um schema também.
- Podemos manipular schema:
    - API de criar index direto quando index ainda não existe
    - API de mapping em si para atualizar um index que já existe.

Listing 4.4 Creating the schema up front

```
PUT employees
{
  "mappings": {
    "properties": {
      "name":{ "type": "text"},
      "age": {"type": "integer"},
      "email": {"type": "keyword"},
      "address":{
        "properties": {
          "street":{ "type":"text" },
          "country":{ "type":"text" }
        }
      }
    }
  }
}
```

The name field is a text data type.
The email field is a keyword type.
The age field is a number type.
Address object in an inner object with further fields
Inner object properties

- No exemplo acima, *address* é um data type **object**, mas não precisa ser explicitado quando contém subcampos.

Listing 4.5 Updating the existing index with additional fields

```
PUT employees/_mapping
{
  "properties":{
    "joining_date":{
      "type":"date",
      "format":"dd-MM-yyyy"
    },
    "phone_number":{
      "type":"keyword"
    }
  }
}
```

_mapping endpoint to update the existing index
The joining date field is a date type.
Expected date format
Phone numbers are stored as is.

Aqui **properties** está no nível root, no exemplo anterior era um campo de *mappings*
ElasticSearch não permite modificar campos existentes, apenas acrescentar mais campos.
    - Previne erros de busca
Uma alternativa a isso é Reindexação:

- i.e. criar um novo index com as especificações desejadas
- Mover os dados do index antigo para o novo, até pode deletar o antigo.
- Tem um API que já faz isso por conta própria

Para evitar problemas com a mudança do nome do index, podemos usar *aliases*.

- Mantém o mesmo nome que pode apontar para index diferentes.

ElasticSearch faz certo type coercion

- Permite inferir transformação do dado inputado um pouquinho para encaixar no tipo aceitável.
  - E.g. número envolto de aspas não dão erro se o data type é numerico.
  - *Type Coercion*

Existem muitos data types, e sempre há novos.

- Types simples:
  - Boolean, text, long, date etc
- Types complexos:
  - Objects, nested, join etc
- Types especializados:
  - Geo_point, ip, date_range, ip_range etc.

Table 4.1 Common data types with examples

Type	Description	Examples
`text`	Represents textual information (such as string values); unstructured text	Movie title, blog post, book review, log message
`integer, long, short, byte`	Represents a number	Number of infectious cases, flights canceled, products sold, book's rank
`float, double`	Represents a floating-point number	Student's grade point average, moving average of sales, reviewer average ratings, temperature
`boolean`	Represents a binary choice: true or false	Is the movie a blockbuster? Are COVID cases on the rise? Has the student passed the exam?
`keyword`	Represents structured text: text that must not be broken down or analyzed	Error codes, email addresses, phone numbers, Social Security numbers
`object`	Represents a JSON object	(In JSON format) employee details, tweets, movie objects
`nested`	Represents an array of objects	Employee address, email routing data, a movie's technical crew

Há vários tipos de text types, types bem específicos.

Text Types

- Texto é decomposto nas suas partes, sem:
  - Pontuação
  - maiúsculas
  - terminações de palavras (stemming)
    - Ficam armazenados numa lista palavra por palavra.
- API _analyze mostra processo de análise.
- Token_count data type
  - Conta número de tokens
  - Field pode ser tanto texto quanto token_count

Listing 4.12 Adding `token_count` as an additional data type to a `text` field

```
PUT tech_books2
{
  "mappings": {
    "properties": {
      "title": {
        "type": "text",
        "fields": {
          "word_count": {
            "type": "token_count",
            "analyzer": "standard"
          }
        }
      }
```

The title field is defined as text data type.

The title field is declared to have multiple data types.

word_count is the additional field.

The type of word_count

It is mandatory to provide the analyzer.

Listing 4.13 Searching for books with a four-word title

```
GET tech_books/_search
{
  "query": {
    "term": {
      "title.word_count": {
        "value": 4
      }
    }
  }
}
```

We are using a term query.

The name of the field

Keyword data type

- Inclui tipos *keyword, contant_keyword, wildcard*
- Keyword é usado para resultado exatos, nunca parciais (e.g. um email)
  - Campo não é analisado pelo analyzer
  - Não é capaz de ser buscado por range
- *Constant_keyword*
  - Keyword já pré-definido no mapping
  - Pode ser omitido ao indexar documentos nesse index
- *Wildcard*
  - Permite realizar buscas com regexs e caracters coringas.

Listing 4.16 A `wildcard` query with a wildcard value

```
GET errors/_search
{
  "query": {
    "wildcard": {
      "description": {
        "value": "*obj*"
      }
    }
  }
}
```

Uses a wildcard query

Searches using wildcards

- **Date Data Type**
  - Dado estruturado, pode ser ordenado, filtrado e agregado.
  - ISO 8601 por padrão

        - Pode ser customizado para permitir múltiplos formatos
    - Permite busca por range

- **Numeric Data Types**
    - **Integers** e **Floats**
        - Vários tipos de cada
            - E.g. long, float, unsigned_long, byte

- **Boolean Data Type**
- **Range Data Type**
    - Limite inferior e superior para um campo.
    - Há vários tipos de range
        - Integer_range, float_range etc.

Listing 4.19 Searching for courses between two dates

```
GET trainings/_search
{
  "query": {
    "range": {
      "training_dates": {
        "gt": "2021-08-10",
        "lt": "2021-08-12"
      }
    }
  }
}
```

We use a range query to search.

Searches for courses between these two dates

```
PUT trainings/_doc/1
{
  "name":"Functional Programming in Java",
  "training_dates":{
    "gte":"2021-08-07",
    "lte":"2021-08-10"
  }
}
```

- **IP Data type**
    - IPv4 ou IPv6

DATA TYPES AVANÇADOS

- **Geo_point data type**
    - Latitude e longitude
    - Permite buscar por *geo_bounding_box*

Table 4.3 Location information-related formats

Format	Explanation	Example
Array	Geopoint represented as an array. Note the order of the geopoint inputs: it takes `lon` and `lat`, not the other way around (unlike the string format—see the next row).	`"address":[0.12,51.5]`
String	Geopoint as string data with `lat` and `lon` inputs.	`"address":"51.5,0.12"`
Geohash	An encoded string formed by hashing the longitude and latitude coordinates. The alphanumeric string points to a place on earth.	`u10j4`
Point	A precise location on a map. Known as well-known text (WKT), a standard mechanism to represent geometrical data.	`POINT(51.5,-0.12)`

- **Object Data Type**
    - É um JSON, tem hierarquia de campos
    - E.g. campo *attachment* com dois subcampos text

Listing 4.26 Searching for emails with an attachment file

```
GET emails/_search
```

Listing 4.26 Searching for emails with an attachment file

```
GET emails/_search
{
  "query": {
    "match": {
      "attachments.filename": "file1.txt"
    }
  }
}
```

    - Têm uma limitação de que objects internos são armazenados como array, não individualmente.
        - E.g. Se eu buscar pelo campo A por um valor e o campo B por outro, vai me retornar resultados mesmo que não seja o mesmo objeto que tenha esses values, vai procurar no array inteiro.
            - Se eu tiver no meu documento um hambúrguer com picles e um hotdog com pimenta, retornará o documento se eu pesquisar por hambúrguer com pimenta.
                - Pode gerar erros, contraintuitivo.
- **Nested Data Type**
    - Resolve exatamente o problema descrito anteriormente
    - Forma especializada de ***object***
    - Deve ser declarado no mapping

- Não existe tipo Array no Elastic Search
    - Mas dá pra armazenar quantos valores forem num dado campo.
    - `"name": "John Doe"` to `"name": ["John Smith", "John Doe"]`
    - Array não pode ter tipos diferentes

- **Flattened Data Type**
    - Previne que todos os subcampos dentro do flattened sejam analisados
        - Todos os subcampos viram keywords.
        - Subcampos não precisam ser declarados no mapping, quaisquer subcampos que apareçam
        - Podemos buscar por qualquer valor de qualquer subcampo ao mesmo tempo sem saber as chaves

Listing 4.33 Indexing a consultation document with doctor notes

```
PUT consultations/_doc/1
{
  "patient_name":"John Doe",
  "doctor_notes":{
    "temperature":103,
    "symptoms":["chills","fever","headache"],
    "history":"none",
    "medication":["Antibiotics","Paracetamol"]
  }
}
```

The flattened field can hold any number of subfields.

All of these fields are indexed as keywords.

Listing 4.34 Searching the `flattened` data type field

```
GET consultations/_search
{
  "query": {
    "match": {
      "doctor_notes": "Paracetamol"
    }
  }
}
```

Searching for a patient's medication

- **Join Data Type**
    - Cria relação pais-filhos
    - Parece com uso em Banco de Dados Relacionais

Listing 4.36 Mapping of the `doctors` schema definition

```
PUT doctors
{
  "mappings": {
    "properties": {
      "relationship":{          <- Declares a property as the join type
        "type": "join",
        "relations":{
          "doctor":"patient"    <- Names of the relations
        }
      }
    }
  }
}
```

The query has two important points to note:

- We declare a `relationship` property of type `join`.
- We declare a `relations` attribute and give the names of the relations (in this case, only one `doctor:patient` relation).

Listing 4.37 Indexing a doctor document

```
PUT doctors/_doc/1
{
  "name":"Dr. Mary Montgomery",
  "relationship":{
    "name":"doctor"          <- The relationship attribute must be one of the relations.
  }
}
```

Listing 4.38 Creating two patients for our doctor

```
PUT doctors/_doc/2?routing=mary          <- Documents must have the routing flag set.
{
  "name":"John Doe",
  "relationship":{                       <- We define the type of relationship in this object.
    "name":"patient",                    <- The document is a patient.
    "parent":1                           <- The patient's parent (doctor) is the document with ID 1.
  }
}

PUT doctors/_doc/3?routing=mary
{
  "name":"Mrs. Doe",
  "relationship":{
    "name":"patient",
    "parent":1
  }
}
```

- Pais e filhos ficam armazenados no mesmo shard
- Desempenho desse tipo de busca não é muito bom
  - Costuma ser melhor evitar e deixar para banco de dados relacionais.

- **Search_as_you_type Data Type**
  - Analyzer cria sub tokens e multi-tokens para tentar buscar com o que o usuário está digitando

Table 4.5 Subfields created automatically by the engine

Fields	Explanation	Examples
`title`	The `title` field is indexed with either a chosen analyzer or the default analyzer if one is not chosen.	If a standard analyzer is used, the title is tokenized and normalized based on the standard analyzer's rules.
`title._2gram`	The `title` field's analyzer is customized with a shingle-token filter. The shingle size is set to 2 on this filter.	Generates two tokens for the given text. For example, the 2-grams for the title "*Elasticsearch in Action*" are `["elasticsearch", "in"], ["in","action"]`.
`title._3gram`	The `title` field's analyzer is customized with a shingle-token filter. The shingle size is set to 3 on this filter.	Generates three tokens for the given text. For example, the 3-grams for the title "*Elasticsearch for Java developers*" are `["elasticsearch","for","java"], ["for","java","developers"]`
`title._index prefix`	The analyzer of `title._3gram` is applied along with an edge n-gram token filter	Generates edge n-grams for the field `title._3grams`. For example, the `_index_prefix` generates the following edge n-grams for the world "Elastic": `[e, el, ela, elas, elast, elasti, elastic]`

Listing 4.42 Searching in `search_as_you_type` and its subfields

```
GET tech_books4/_search
{
  "query": {
    "multi_match": {
      "query": "in",
      "type": "bool_prefix",
      "fields": ["title","title._2gram","title._3gram"]
    }
  }
}
```

- **Multiple Types**
  - Campos podem ser multi-tipos no Elastic Search, se declararmos isso no mapping
  - Pode ser *Text*, *keyword* e *completion* ao mesmo tempo

Listing 4.43 Schema definition with a multi-typed field

```
PUT emails_multi_type
{
  "mappings": {
    "properties": {
      "subject":{
        "type": "text",
        "fields": {
          "kw":{ "type":"keyword" },
          "comp":{ "type":"completion" }
        }
      }
    }
  }
}
```

The text type

The subject is also a keyword.

The subject is a completion type, too.

Summary

- Every document consists of fields with values, and each field has a data type. Elasticsearch provides a rich set of data types to represent these values.
- Elasticsearch consults a set of rules when indexing and searching data. These rules, called mapping rules, let Elasticsearch know how to deal with the varied data shapes.
- Mapping rules are formed by either dynamic or explicit mapping processes.

- Mapping is a mechanism for creating field schema definitions up front. Elasticsearch consults the schema definitions while indexing documents so the data is analyzed and stored for faster retrieval.
- Elasticsearch also has a default mapping feature: we can let Elasticsearch derive the mapping rather than providing it explicitly ourselves. Elasticsearch determines the schema based on the first time it sees a field.
- Although dynamic mapping is handy, especially in development, if we know more about the data model, it is best to create the mapping beforehand.
- Elasticsearch provides a wide range of data types for text, Booleans, numerical values, dates, and so on, stretching to complex fields like joins, completion, geopoints, nested, and others.