04

Índices, tipos, shards e replicas

O nosso projeto

Nesse curso vamos construir uma base de dados com informações sobre pessoas. Queremos habilitar buscas e agrupamentos por diversos atributos, além de buscas por sinônimos. A busca deverá ser o mais agradável possível para o usuário final, logo queremos suportar a busca ao estilo Google, com destaque para os termos buscados no resultado, além da clássica busca estruturada.

Assim como todo projeto que envolve armazenamento de dados, precisamos conhecer muito bem como interagir com nosso repositório de dados. Precisamos saber também como modelar nossos dados de modo a atender nossas necessidades de negócio.

Analogia entre ElasticSearch e um banco de dados relacional

Antes de começarmos nosso projeto para valer, precisamos conhecer um pouco mais sobre o ElasticSearch. No capítulo anterior utilizamos a API Rest para criar documentos. Vale lembrar que não utilizamos nenhum comando prévio para criar o índice ou tipo, apenas criamos documentos para, em seguida, fazermos algumas buscas básicas.

Se olharmos cuidadosamente o resultado da primeira operação de POST executada, vamos notar o seguinte:

```
{
    "_index": "catalogo",
    "_type": "pessoas"
}
```

Mas o que é um *index* e um *type* exatamente? Podemos fazer uma analogia bem simples com os bancos de dados relacionais para nos ajudar a entender melhor o que estes atributos significam.

Banco Relacional	ElasticSearch
Instância	Index
Tabela	Type
Schema	Mapping
Tupla	Documento
Coluna	Atributo

Pela analogia com banco de dados, é natural chegarmos à conclusão que um índice pode ter diversos tipos. Aplicando a analogia ao que criamos no capítulo anterior, podemos dizer que *criamos a tabela pessoas no banco de dados catalogo*.

Cuidado! O termo *índice* utilizado no mundo de bancos de dados relacionais, onde os registros das tabelas são organizados de forma a diminuir o tempo da busca em estruturas de dados apropriadas, não possui analogia ou correspondência com o termo *índice* utilizado no mundo do ElasticSearch. Basta pensar que um índice no ElasticSearch está para um *database* no MySQL.

A analogia também é válida para os comandos SQL do banco de dados com algumas pequenas ressalvas como mostrado a seguir.

Entendendo HEAD e GET

Acessando `HEAD /index/type/id` é equivalente ao comando:

```
select 1 from TYPE where id = ID;
```

Verifica, sem retornar o conteúdo, se o documento cujo identificador é **ID** existe para o tipo **TYPE** no índice *index*. Caso o documento exista, o código HTTP **`OK 200`** é retornado, caso contrário, o código HTTP **`NOT FOUND 404`** é retornado.

**Importante: /index/type/id é um identificador único de um documento em um *cluster* ElasticSearch. Esta tripla é conhecida como Unique Resource Identifier (URI, identificador único de recurso). **

`GET /index/type/id` é equivalente ao comando:

```
select * from TYPE where id = ID;
```

Localiza o documento e devolve o documento, caso ele exista, ou *`* NOT FOUND 404`*, caso contrário.

Ainda que o ElasticSearch seja orientado a documentos, é possível retornar apenas uma parte do documento. Para tal, basta adicionar `?_source=<atributo 1>,<atributo 2>`. Note que podemos ainda utilizar o parâmetro *pretty* para retornar o documento formatado. Exemplo: `GET /catalogo/pessoas/1?pretty&_source=interesses`.

Os métodos DELETE, PUT e POST

Uma requisição `DELETE /index/type/id` é Equivalente ao comando:

```
delete from TYPE where id = ID;
```

Remove o documento do tipo caso ele exista, ou **`NOT FOUND 404`**, caso contrário.

PUT /index/type/id

Aqui temos uma pequena ressalva em relação aos comandos SQL. A rigor, o comando PUT significa *coloque o documento sob o localizador /index/type/id*. Repare que *id*, assim como nos comandos anteriores é obrigatório. Poderíamos pensar então que o comando PUT é uma espécie de:

```
insert into TYPE (id, atributo1, atributo2) values (ID, valor1, valor2);
```

Contudo, também podemos utilizar o comando PUT para substituir um documento. Neste caso, o comando PUT funciona como uma espécie de:

```
update TYPE set atributo1 = valor1, atributo2 = valor2 where id = ID;
```

POST /index/type

Aqui temos mais uma pequena ressalva em relação aos comandos SQL. A rigor, o comando POST significa *crie um documento sob /index/type*. Repare que, diferentemente do comando PUT e dos outros comandos, id não é obrigatório. Poderíamos pensar então que o comando POST é uma espécie de insert que utiliza *sequences* ou um algum tipo de auto incremento. Contudo, também podemos utilizar o comando POST para atualizar um documento existente assim como o comando PUT. A diferença é que POST nos permite atualização parcial de documentos. Por exemplo:

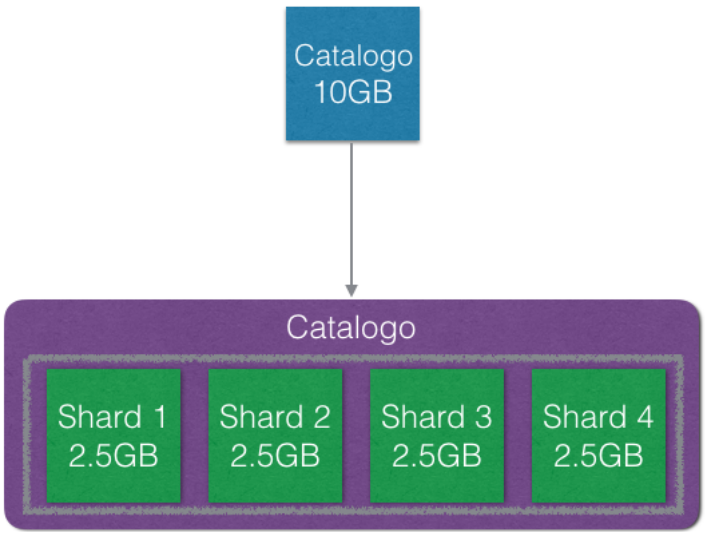
```
POST /catalogo/pessoas/1/_update
{
    "doc": {
        "nome": "João Pedro"
    }
}
```

Note que, neste caso, o uso de *"doc"* é obrigatório.

** Importante: Uma vez que um documento é criado em uma instância do ElasticSearch, este documento torna-se imutável. No caso de uma atualização a um documento existente, por exemplo como fizemos com o método POST, uma nova versão do documento é criada. Se repararmos bem nas respostas recebidas, notaremos os atributos *_created* e *_version*. A resposta para a criação de um novo documento possui `_created = true` e `_version = 1` . A resposta para atualizações possui `_created = false` e `_version` será a versão anterior do documento acrescida de 1. **

Shards e Réplicas

Shard é um termo que passou a ser muito utilizado quando pensamos em particionamento horizontal de dados. Para colocar de modo mais claro, pense que temos um volume muito grande de dados, tão grande, que é inviável em armazená-lo em um único local. Nosso instinto de bom engenheiro de software nos diz que deveríamos *quebrar* este grande volume de dados em blocos menores, já que fica mais fácil de operar com blocos menores durante operações como cópia, remoção ou relocação. Cada pedacinho resultante desta quebra é o que chamamos de *shard*.



Shards são de altíssima importância quando pensamos em escalabilidade horizontal. Podemos quebrar grandes blocos de dados em pedaços menores e distribuí-los entre diferentes máquinas em um *cluster*. Dado o número de shards, podemos usar uma função de espalhamento (*hashing*) para definir em qual *shard* um determinado documento deve ser armazenado. Isso é exatamente o que o ElasticSearch faz (bem, o processo na prática é um pouco mais complexo, mas a

ideia é a mesma). A escalabilidade horizontal para o volume de dados que foi comentada no capítulo anterior é implementada com o uso de *shards*.

Como temos diversas *shards,* podemos criar cópias, chamadas de réplicas, de uma mesma *shard* e armazená-las em outras máquinas. Isso funciona como uma espécie de backup dos dados, com uma pequena ressalva: réplicas são atualizadas constantemente e podem ser utilizadas para leitura durante a execução de consultas. Existem dois tipos de *shards* no ElasticSeach:

- *Shard* primária (*primary shards*): é a *shard* onde as operações de escrita como criação, atualização ou remoção de um documento acontece primeiro.
- *Shard* réplica (*replica shard*): é a *shard* que, uma vez que a operação de escrita tenha sido concluída com sucesso na sua respectiva *shard* primária, recebe a mesma operação para que ela seja replicada. A operação só será confirmada para o cliente quando todas as réplicas confirmarem a replicação. Logo, quando recebemos o HTTP OK para uma operação de escrita, sabemos que a informação esta segura em todas as réplicas.

** Importante: O número de *shards* é definido no momento da criação do índice e não pode ser alterado. O número de réplicas também é definido no momento da criação do índice, porém pode ser alterado com o passar do tempo. É muito importante escolher bem o número de *shards* durante a criação do índice. A escolha deste número depende do volume de informações que queremos armazenar nas *shards*. Em geral, queremos *shards* com alguns gigabytes.**

O que aprendemos?

- O que é um índice e um tipo.
- Como utilizar a API Rest para operações de criação, atualização, remoção e verificação de existência de documentos no ElasticSearch.
- A analogia entre ElasticSearch e um banco de dados relacionais.
- O que são shards, réplicas e qual a sua importância.