53 45 52 45 49 20 46 49 45 4c 20
41 4f 53 20 50 52 45 43 45 49 54
4f 53 20 44 41 20 48 4f 4e 52 41
20 45 20 44 41 20 43 49 c3 8a 4e
43 49 41 2c 20 50 52 4f 4d 4f 56
45 4e 44 4f 20 4f 20 55 53 4f 20
45 20 4f 20 44 45 53 45 4e 56 4f
4c 56 49 4d 45 4e 54 4f 20 44 41
20 49 4e 46 4f 52 4d c3 81 54 49
43 41 20 45 4d 20 42 45 4e 45 46
c3 8d 43 49 4f 20 44 4f 20 43 49
44 41 44 c3 83 4f 20 45 20 44 41
20 53 4f 43 49 45 44 41 44 45 2e

RESIDÊNCIA DE SOFTWARE

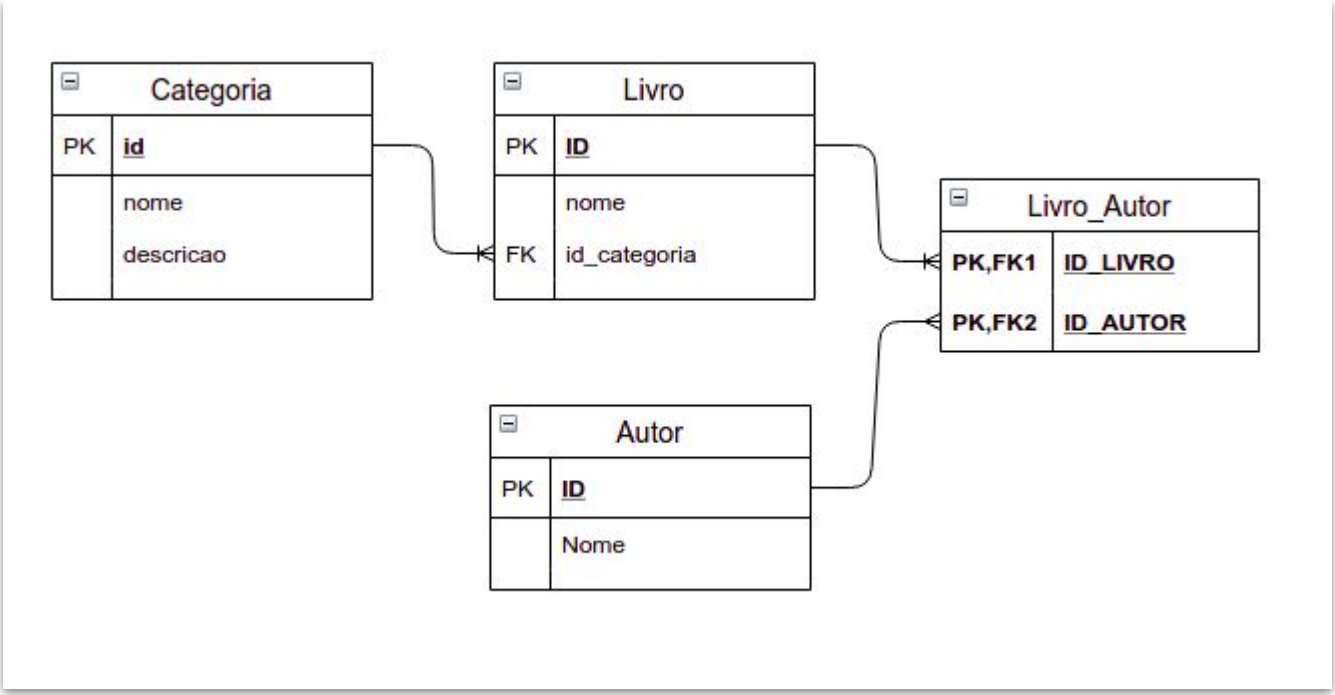
CAPACITAR
TREINAR
EMPREGAR
TRANSFORMAR

Firjan SENAI

serratec
Parque Tecnológico Região Serrana

Java - Web - Spring Boot
05 - Trasações, Relacionamento, Consultas e Swagger

Relacionamento



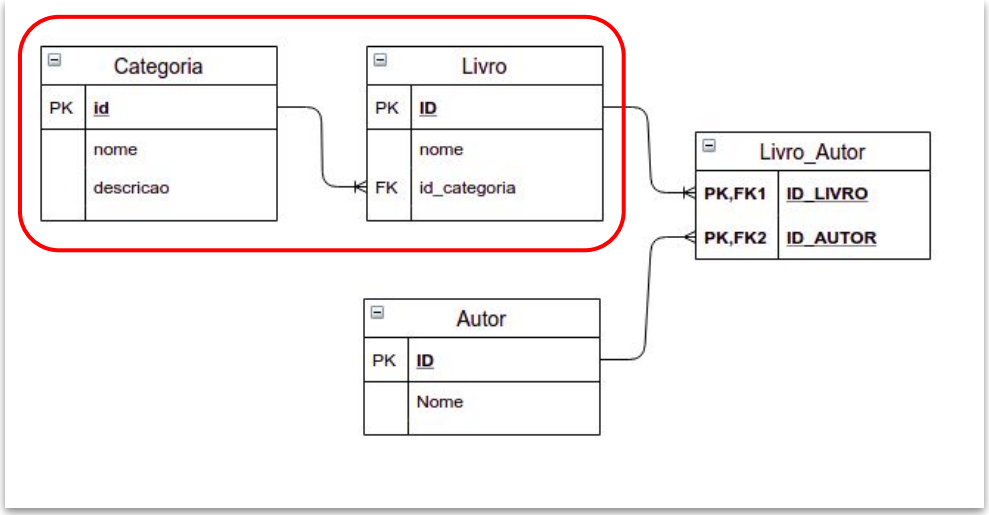
One To Many / Many To One

Entidade Livro

```
@ManyToOne()
@JoinColumn(name="categoria_id",
            referencedColumnName = "id")
private Categoria categoria;
```

Entidade Categoria

```
@OneToMany(mappedBy="categoria")
private Set<Livro> livros;
```



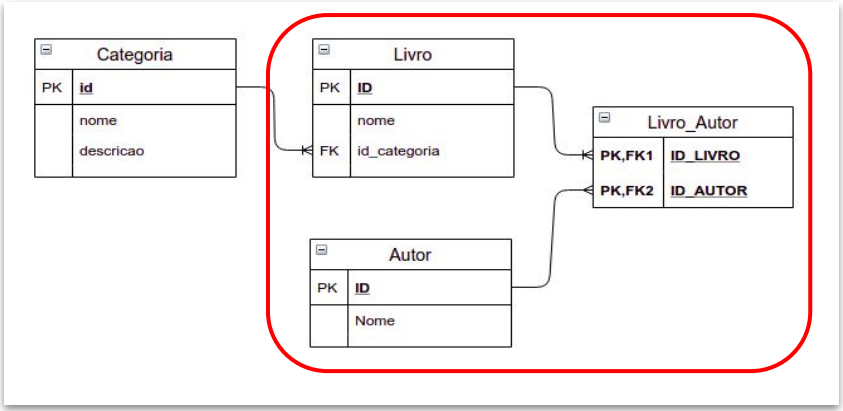
Many to Many

Entidade Livro

```
@ManyToMany(cascade = CascadeType.ALL)
@JoinTable(name = "livro_autor",
     joinColumns = @JoinColumn(name = "livro_id", referencedColumnName = "id"),
     inverseJoinColumns = @JoinColumn(name = "autor_id", referencedColumnName = "id"))
private Set<Autores> autores;
```

@JoinTable - tabela que será utilizada no relacionamento muitos para muitos

- joinColumns - colunas na tabela de ligação que são relacionadas com a tabela principal
- inverseJoinColumns - - colunas na tabela de ligação que são relacionadas com a tabela secundária (no caso Autor)



One to One

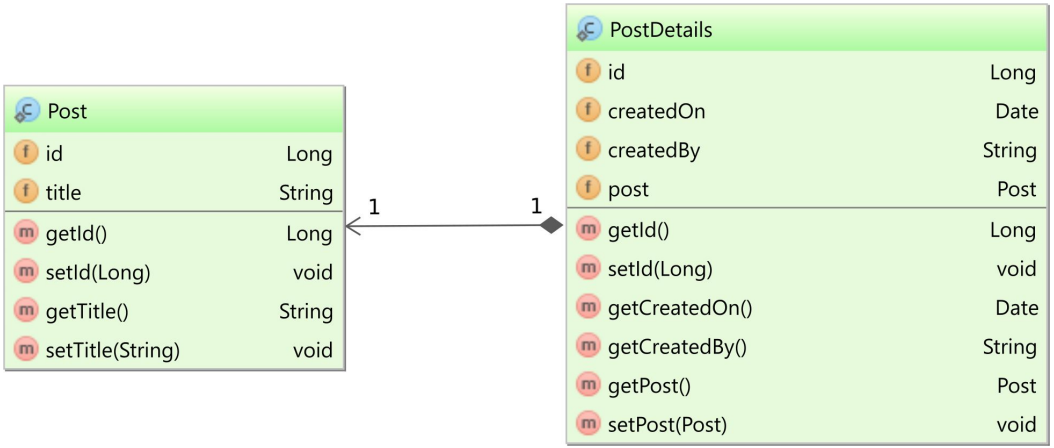
```
@Entity(name = "PostDetails")
@Table(name = "post_details")
public class PostDetails {
        ...

    @OneToOne
    @JoinColumn(name = "post_id")
    private Post post;

        ...
}

@Entity(name = "Post")
@Table(name = "post")
public class Post {
        ...
    @OneToOne(mappedBy = "post")
    private PostDetails details;

        ...
}
```



JPQL - Join / Join fetch

Supondo o mapeamento e a query abaixo:

```
@OneToMany(targetEntity=Livro.class, mappedBy="categoria", cascade=CascadeType.ALL,
           fetch = FetchType.LAZY)
private Set<Livro> livros;

@Query("SELECT C FROM Categoria C where C.id=:id”)
Categoria findByID (@Param("id") Integer id);
```

Ao executar o código:

```
Categoria categoria = categoriaRepository.findById(10);
for(Livro livro: categoria.getLivros()){
    System.out.println(livro.getTitulo());
}
```

O Spring/Hibernate executará a consulta a baixo quando o método findById for executado.

```
Select C.id, C.nome from categoria C;
```

E quando o categoria.getLivros() for executado ele executará a seguinte

```
select L.id, L.titulo, L.autor, L.data_publicacao from livro L where L.categoria_id = 10
```

JPQL - Join / Join fetch

Mudando a query

```
@Query("SELECT C FROM Categoria C join fetch C.livros where C.id=:id”)
Categoria findByID (@Param("id") Integer id);
```

Ao executar o código:

```
Categoria categoria = categoriaRepository.findById(10);
for(Livro livro: categoria.getLivros()){
    System.out.println(livro.getTitulo());
}
```

O Spring/Hibernate executará a consulta a baixo quando o método findById for executado.

```
select C.id, C.nome, L.id, L.titulo, L.autor, L.data_publicacao from categoria C join
livro L on L.categoria_id = C.categoria_id;
```

Agregação / Interfaces

Contar total de Livros por autor:

Criar uma interface para armazenar o resultado (o Spring irá criar, em tempo de execução, um objeto que implemente esta interface)

```
public interface LivrosPorAutor {
   String getNomeAutor();
   Int getTotal();
}
```

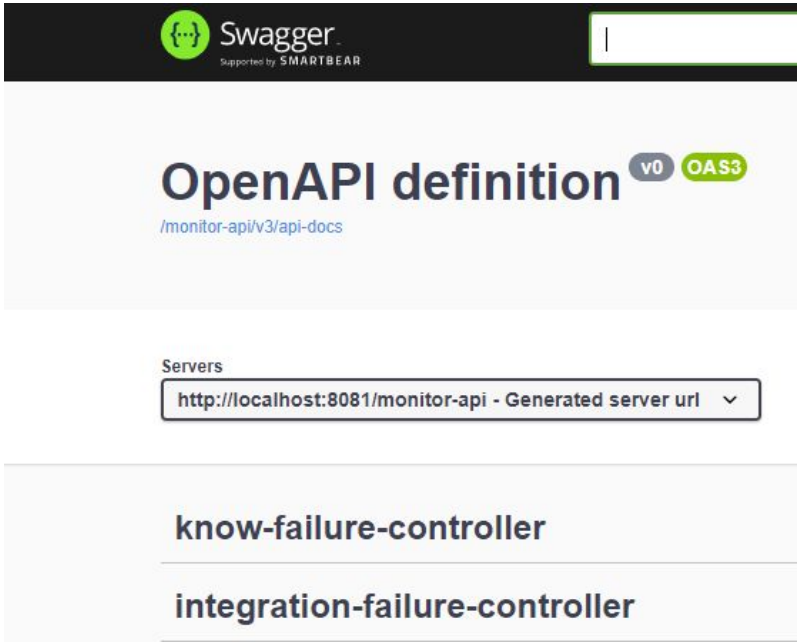
Criar o método no Repositório com a JPQL que realiza o join e o group by

```
@Query("select A.nome nomeAutor, count(L)
            from Livro L join L.autores A group by A.nome order by A.nome")
List<LivrosPorAutor> buscaLivrosPorAutor();
```

Swagger - OpenAPI

- Adicionar dependência do OpenAPI

```
<dependency>
      <groupId>org.springdoc</groupId>
      <artifactId>springdoc-openapi-ui</artifactId>
</dependency>
```



Mais informações https://www.baeldung.com/spring-rest-openapi-documentation

Exercício

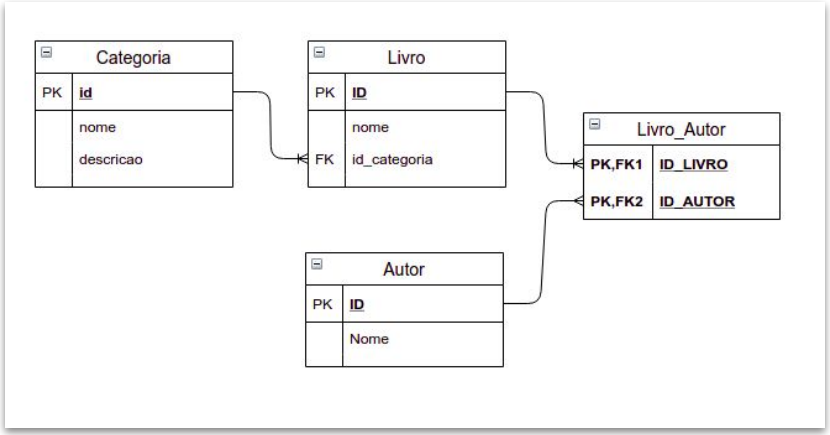
Criar um sistema com as seguintes APIs

- Entidade categoria:
  { “id”: integer, “nome”:string, “descricao”:string}
  GET /categoria/
  GET /categoria/{ID}
  POST /categoria
  PUT /categoria/{ID}
  DETETE /categoria/{ID}

- Entidade Autor
  { “id”:integer, “nome”:string}
  GET /autor/
  GET /autor/{ID}
  POST /autor
  PUT /autor/{ID}
  DELETE /autor/{ID}

- Documentar a API com Swagger

- Entidade Livro:
  {“id”: integer, “nome”:string,
  “categoria”:ENTIDADE_CATEGORIA,
  “Autores”: [ENTIDADE_AUTOR,
  ENTIDADE_AUTOR,...]}
  GET /autor/
  GET /autor/{ID}
  POST /autor
  PUT /autor/{ID}
  DELETE /autor/{ID}



Exercício

Bonus:

- Ao excluir uma categoria, se ela já estiver associada há um livro, enviar mensagem de erro
- Ao excluir uma autor, se ele já estiver associado há um livro, enviar mensagem de erro
- Ao cadastrar ou alterar um livro, se a categoria ou um dos autores não existir, deve enviar mensagem de erro
- Incluir a documentação com Swagger UI