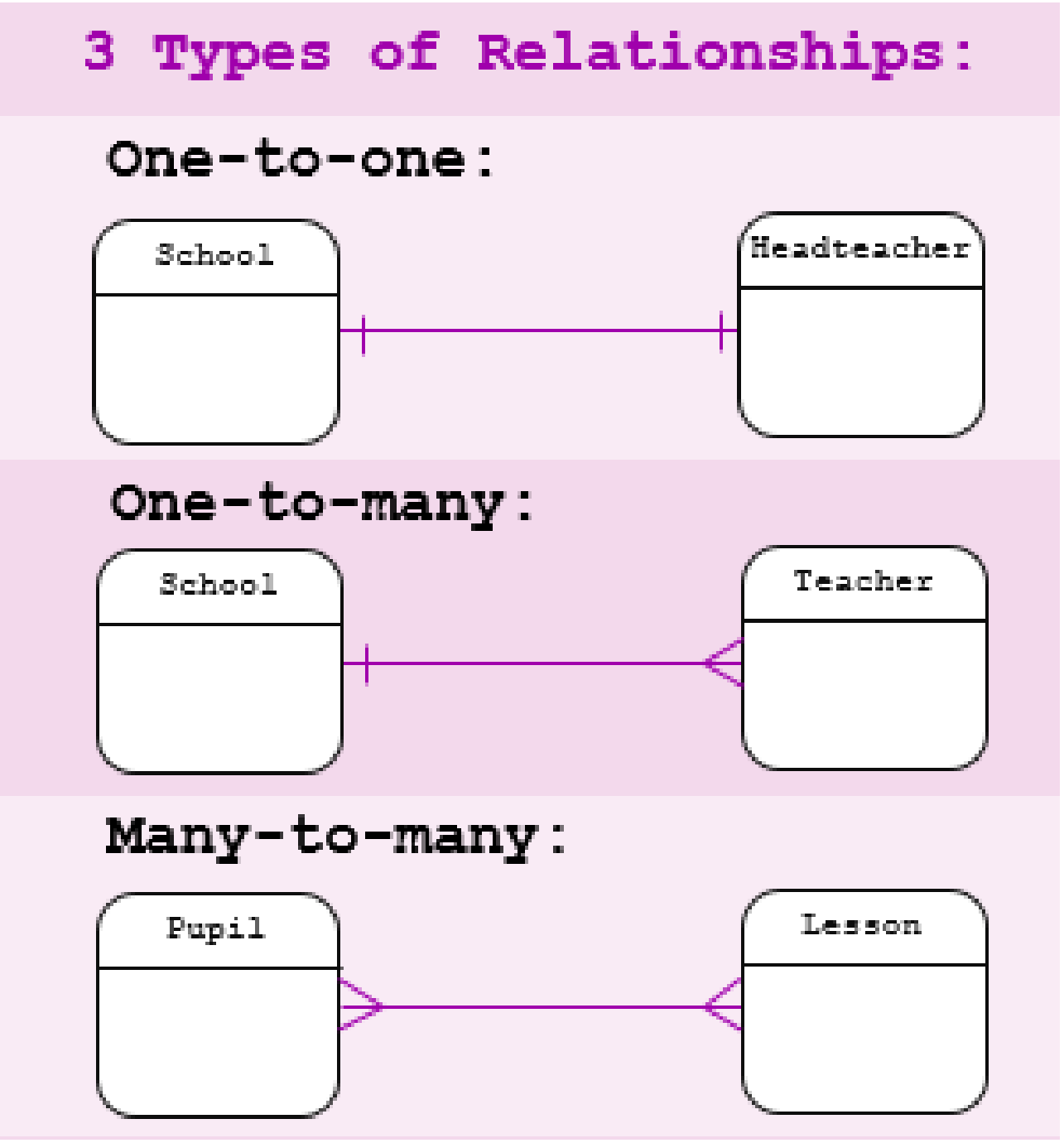
TÓPICO 17 - CONSULTAS AVANÇADAS

Disciplina de Backend - Professor Ramon Venson - SATC 2026.1



Relacionamentos

Entidades em um banco de dados relacional podem possuir relacionamentos entre si.

Esses relacionamentos podem ser de diferentes tipos:

- Um-para-um (1:1);
- Um-para-muitos (1:N);
- Muitos-para-muitos (N:N);

Exemplo de Relacionamento

No Spring, podemos representar esses relacionamentos usando anotações do JPA.

```
@Entity
public class Estudante {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    @OneToMany(mappedBy = "estudante", cascade = CascadeType.ALL)
    private List<Disciplina> disciplinas;
}
```

Isso permite que o JPA entenda como as entidades estão relacionadas e como realizar consultas que envolvam essas relações.

Anotações de Relacionamento

Anotação	Descrição
`@OneToOne`	Define um relacionamento um-para-um entre duas entidades.
`@OneToMany`	Define um relacionamento um-para-muitos entre duas entidades
`@ManyToOne`	Define um relacionamento muitos-para-um entre duas entidades.
`@ManyToMany`	Define um relacionamento muitos-para-muitos entre duas entidades.
`@JoinColumn`	Especifica a coluna que será usada para o relacionamento.
`@JoinTable`	Especifica a tabela intermediária para relacionamentos muitos-para-muitos.

Mapeamento Bidirecional

O mapeamento bidirecional permite que ambas as entidades em um relacionamento possam referenciar uma à outra.

```
@Entity
public class Disciplina {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    @ManyToOne
    @JoinColumn(name = "estudante_id")
    private Estudante estudante;
}
```

Problemas com Mapeamento Bidirecional

O problema do mapeamento bidirecional é que ele pode causar problemas de recursão infinita durante a serialização para JSON.

Dessa forma, é comum usar anotações como `@JsonManagedReference` e `@JsonBackReference` para evitar esses problemas.

```
    @OneToMany(mappedBy = "estudante", cascade = CascadeType.ALL)
    @JsonManagedReference
    private List<Disciplina> disciplinas;
```

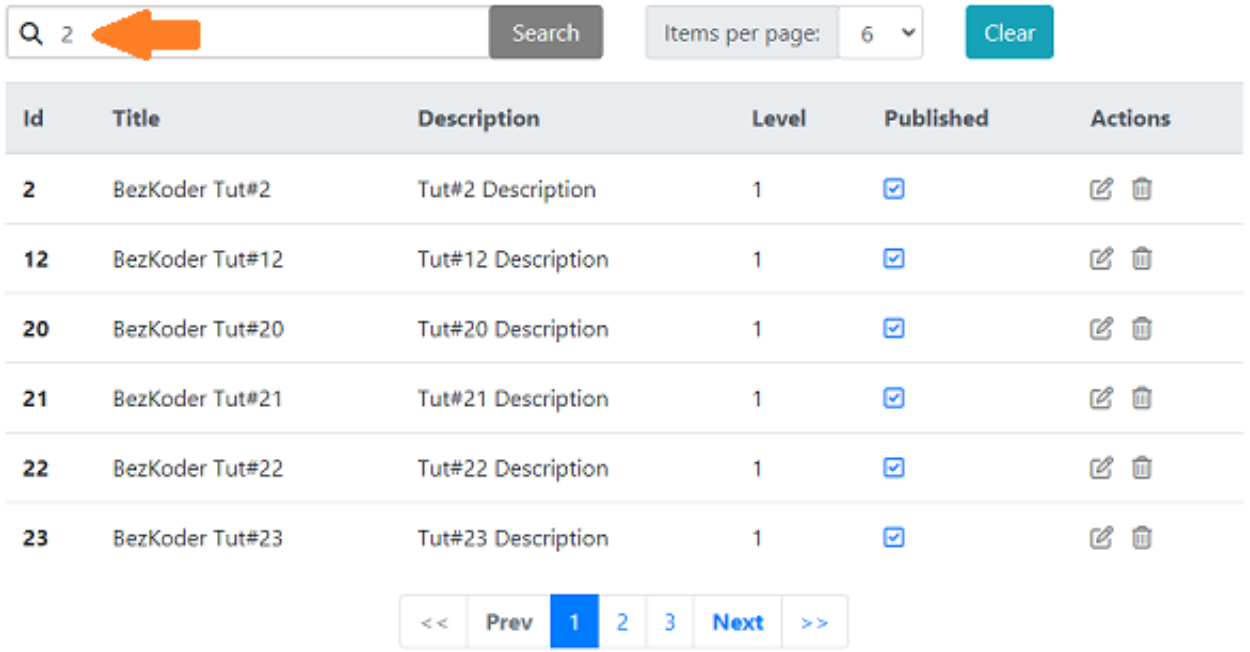
Boas Práticas

- Use mapeamento bidirecional apenas quando necessário;
- Considere usar DTOs para evitar problemas de serialização;
- Utilize `@JsonIgnore` para evitar serialização de campos que não são necessários na resposta;
- Use o `fetch = FetchType.LAZY` para carregar relacionamentos apenas quando necessário;
- Teste suas entidades e relacionamentos para garantir que funcionem conforme o esperado;

Paginação e Ordenação

A paginação é usada para evitar desperdício de tráfego de saída. Ordenação garante que os dados sejam apresentados em uma ordem específica antes da sua paginação.

Esses recursos são **usados especialmente com consultas do tipo** `GET`



Paginação

Para realizar a paginação de uma requisição, vamos criar um `PageRequest` (que implementa a interface `Pageable`).

```
Pageable page = PageRequest.of(paginaAtual, tamanhoPagina);
Page<Animal> list = animalRepository.findAll(page);
```

No método, podemos passar a página atual (começando do 0) e o tamanho da página (quantos resultados em cada página).

Paginação no Controllerlista

No controlador, vamos precisar solicitar do cliente a página e o número de resultados que ele pretende acessar, repassando essa configuração ao serviço.

A paginação é feita usando de *Query Strings* e, portanto, o `@RequestParam`. Ex.:

`http://localhost:8080/animais?pagina=5&resultados=5`

```
@GetMapping
@ResponseStatus(HttpStatus.OK)
public List<AnimalDto> getAllAnimaisByPageAndSort(
        @RequestParam(defaultValue = "0") Integer pagina,
        @RequestParam(defaultValue = "10") Integer resultados) {
    return animalService.getAllAnimais(pagina, resultados);
}
```

> O uso do `defaultValue` serve para tornar o uso facultativo na `Query String`

Paginação no Service

No service, vamos criar um objeto `Pageable` com as configurações de paginação.

```
public List<AnimalDto> getAllAnimais(Integer pagina, Integer resultados) {
    // Gera uma configuração de paginação
    Pageable page = PageRequest.of(pagina, resultados);
    // Consulta no repositório
    Page<Animal> list = animalRepository.findAll(page);
    // Conversão de todos os objetos da página para um DTO
    return list.stream().map(animalMapper::toDto).toList();
}
```

Ordenação

Podemos adicionar à configuração de paginação uma ordenação para que a lista de retorno esteja organiza por um ou mais atributos:

```
Pageable page = PageRequest.of(
    paginaAtual,
    tamanhoPagina,
    Sort.by('nome').ascending() // Ordena por nome em ordem ascendente
);

Page<Animal> list = animalRepository.findAll(page);
```

Os métodos `ascending` e `descending` podem definir a direção da ordenação.

> A ordenação é realizada antes da paginação pelo Spring Data

Podemos utilizar também uma lista com mais de um atributo para ordenação, porém, precisaremos converter/mapear a lista para o tipo de dado adequado: `Sort.Order`:

```
// List<String> sortList -> ["nome", "idade"]
List<Sort.Order> ordemLista = sortList.stream().map((sort) -> {
    return new Sort.Order(Sort.Direction.ASC, sort);
}).toList();

Pageable page = PageRequest.of(
    paginaAtual,
    tamanhoPagina,
    Sort.by(ordemLista)
);

Page<Animal> list = animalRepository.findAll(page);
```

Ordenação no Controller

No controller, vamos receber mais uma Query String usando o @RequestParam. A URL da requisição será algo parecido com isso: `http://localhost:8080/animais?pagina=0&resultados=5&ordenar=idade,nome`

```
@GetMapping
@ResponseStatus(HttpStatus.OK)
public List<AnimalDto> getAllAnimaisByPageAndSort(
        @RequestParam(defaultValue = "-1") Integer pagina,
        @RequestParam(defaultValue = "-1") Integer resultados,
        @RequestParam(defaultValue = "") List<String> ordenar) {
    return animalService.getAllAnimais(page, results, ordenar);
}
```

Ordenação no Service

Por fim, nosso service ficará da seguinte maneira:

```
public List<AnimalDto> getAllAnimais(Integer pagina, Integer resultados, List<String> ordenar ) {
    // Mapeia a lista de strings para Sort.Order
    List<Sort.Order> ordemFinal = sortList.stream()
    .map((ordenar) -> {
        return new Sort.Order(Sort.Direction.ASC, sort);
    }).toList();
    // Cria a configuração de paginação
    Pageable page = PageRequest.of(
        pagina,
        resultados,
        Sort.by(ordemFinal)
    );
    // Faz a requisição
    Page<Animal> list = animalRepository.findAll(page);
    // Converte a página para um DTO
    return list.stream().map(animalMapper::toDto).toList();
}
```