

PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli



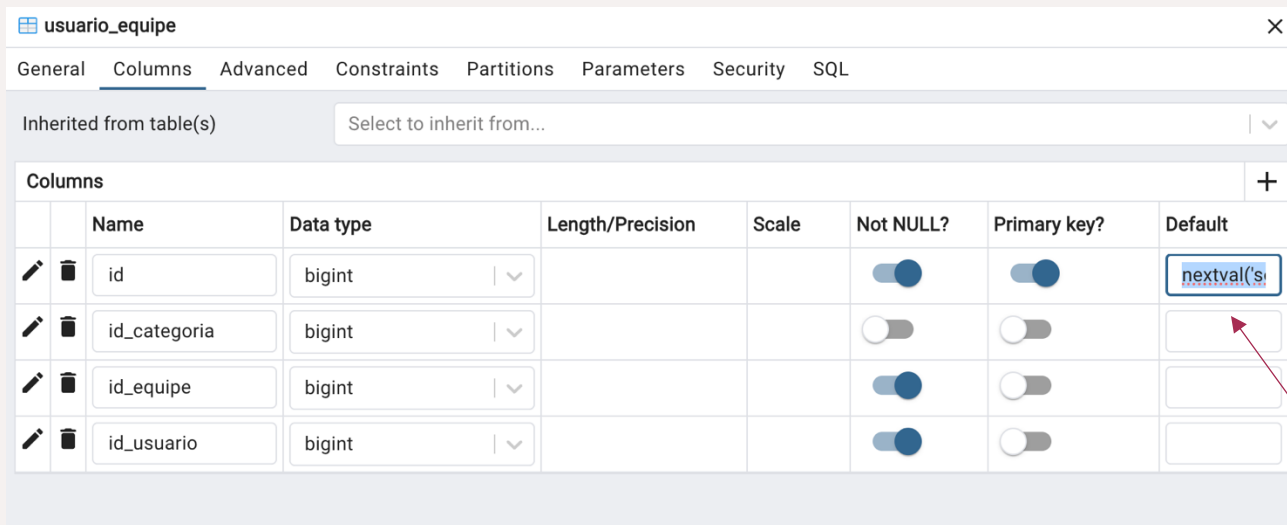
The top of the slide features a dark red background with a series of overlapping geometric patterns. These include concentric circles, semi-circles, and radial lines, all rendered in a lighter shade of red, creating a modern, abstract design.

USUÁRIO EQUIPE

Contexto

Aula 07









BANCO DE DADOS



usuario_equipe

General Columns Advanced Constraints Partitions Parameters Security SQL

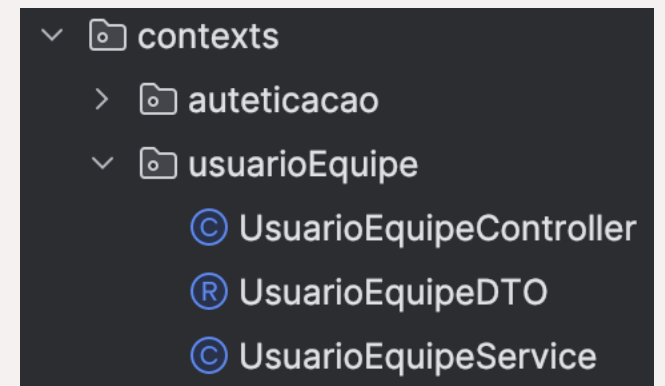
Inherited from table(s) Select to inherit from...

| | Name | Data type | Length/Precision | Scale | Not NULL? | Primary key? | Default |
|---|--------------|-----------|------------------|-------|-------------------------------------|-------------------------------------|---------------|
|   | id | bigint | | | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | nextval('s... |
|   | id_categoria | bigint | | | <input type="checkbox"/> | <input type="checkbox"/> | |
|   | id_equipe | bigint | | | <input checked="" type="checkbox"/> | <input type="checkbox"/> | |
|   | id_usuario | bigint | | | <input checked="" type="checkbox"/> | <input type="checkbox"/> | |

- Cada sequência criada precisa ser associada a respectiva tabela que pertence;
- No valor default de id (tabela usuario_equipe), adicione a sequence correspondente, por meio do comando:
`nextval('seq_usuario_equipe')`

CRIANDO O CONTEXTO

- Pacote exclusivo dos contextos
- Dentro dos contextos, criar o pacote de usuarioEquipe
- Esse contexto deve conter operações básicas para:
 - Inserir várias equipes para o usuário
 - Remover usuário de uma determinada equipe
 - Mostrar os usuários e suas respectivas equipes



ENTIDADE USUÁRIO

- O QUE MUDA?

```
48     @OneToMany
49     @JoinTable(name = "usuario_equipe", schema = "public",
50               joinColumns = @JoinColumn(name = "id_usuario"),
51               inverseJoinColumns = @JoinColumn(name = "id_equipe"))
52     @JsonManagedReference
53     private Collection<Equipe> equipes;
```

- A tabela usuario_equipe serve de ligação entre as tabelas Usuario e Equipe;
- Para nós, interessa saber em quais equipes o usuário está inserido – o nome das equipes;
- Mapeamos aqui uma tabela de ligação (usuario_equipe) para que seja obtida todas as equipes (MANY) que o usuário está inserido.

ENTIDADE USUARIOEQUIPE

```
8  @Data 1 usage
9  @Entity
10 @Table(name = "usuario_equipe", schema = "public")
11 public class UsuarioEquipe implements Serializable {
12
13     public static final long serialVersionUID = -4543293329387972832L;
14
15     @Id
16     @SequenceGenerator(
17         name = "SEQ-USUARIO-EQUIPE",
18         sequenceName = "public.seq_usuario_equipe",
19         allocationSize = 1)
20     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQ-USUARIO-EQUIPE")
21     private Long id;
22
23     @OneToOne
24     @JoinColumn(name = "id_usuario", referencedColumnName = "id")
25     private Usuario usuario;
26
27     @ManyToOne
28     @JoinColumn(name = "id_equipe", referencedColumnName = "id")
29     private Equipe equipe;
30
31     @ManyToOne
32     @JoinColumn(name = "id_categoria", referencedColumnName = "id")
33     private Categoria categoria;
34
35 }
```

- Embora não acessada diretamente, a entidade de ligação precisa estar explícita no código da POO.

DTO

- Com base em um determinado usuário, retorna os dados de e-mail, perfil e equipes que está inserido

```
public record UsuarioEquipeDTO( 4 usages
    String nome, no usages
    String email, no usages
    String perfil, no usages
    List<String> equipes no usages
) implements Serializable {
```

```
@Serial
public static final long serialVersionUID = -5438278624684826842L;

public static UsuarioEquipeDTO valueOf(Usuario usuario) { 1 usage
    if (usuario != null) {
        //verificar se existem equipes
        if(usuario.getEquipes() != null && !usuario.getEquipes().isEmpty()) {
            //preenche a lista de equipes que o usuário está inserido
            List<String> equipes = new ArrayList<>();
            usuario.getEquipes().forEach(equipe -> equipes.add(equipe.getEquipe()));
            return new UsuarioEquipeDTO(
                usuario.getNome(),
                usuario.getEmail(),
                usuario.getPerfil().getPerfil(),
                equipes
            );
        } else return null;
    } else return null;
}
```

SERVICE

- Deve tanto adicionar usuários nas equipes quanto remover
- Por segurança, deve sempre verificar se o usuário informado está na base de dados
- Um único método faz a inserção e remoção de equipes do usuário

```
13 @Service 1 usage
14 public class UsuarioEquipeService {
15
16     @Autowired
17     UsuarioService usuarioService;
18
19     public Usuario addEquipes(String email, List<EquipeDTO> equipesDTO) { 1 usage
20         Usuario usuario = this.findUsuarioByEmail(email);
21         if(usuario != null && equipesDTO != null && !equipesDTO.isEmpty()) {
22             List<Equipe> equipes = new ArrayList<>();
23             equipesDTO.forEach(equipe -> equipes.add(EquipeDTO.toEquipe(equipe)));
24             usuario.setEquipes(equipes);
25             return usuarioService.cadastrar(usuario);
26         }
27         return null;
28     }
29
30
31
32     private Usuario findUsuarioByEmail(String email) { 1 usage
33         return usuarioService.findByEmail(email);
34     }
}
```

- Com o mapeamento feito entre usuário e equipe, basta que a lista das equipes de um usuário seja atualizada para, automaticamente, atualizar a tabela de vínculo usuario_equipe.
- Note que a lista de equipes de um usuário SEMPRE é reconstruída no código (linhas 23 e 24).

CONTROLLER

- Endpoint para listar as equipes disponíveis e adicionar equipes aos usuários cadastrados;
- O endpoint /all repete código que poderia já estar inserido na classe EquipeController

```
13 @RestController
14 @RequestMapping("usuarioEquipe/")
15 public class UsuarioEquipeController {
16
17     @Autowired
18     UsuarioEquipeService usuarioEquipeService;
19
20     @Autowired
21     EquipeService equipeService;
22
23     @GetMapping("all")
24     public @ResponseBody ResponseEntity<List<EquipeDTO>> buscarEquipes() {
25         List<Equipe> equipes = equipeService.buscarTodas();
26         return ResponseEntity.ok().body(EquipeDTO.valueAll(equipes));
27     }
28
29     @PostMapping("add/{email}")
30     public @ResponseBody ResponseEntity<UsuarioEquipeDTO> addEquipes(@PathVariable String email,
31                                                                    @RequestBody List<EquipeDTO> equipes) {
32         Usuario usuario = usuarioEquipeService.addEquipes(email, equipes);
33         return ResponseEntity.ok().body(UsuarioEquipeDTO.valueOf(usuario));
34     }
35
36 }
```

TESTE

- Inserindo Equipes para a nossa Usuária via Postman

The screenshot shows the Postman interface for a REST client. The top bar indicates the request is a **POST** to `http://localhost:8080/HelpDesk/api/usuarioEquipe/add/karina.almeida@fatec.br`. The **Params** tab is active, showing a table for Query Params.

| Key | Value | Description |
|-----|-------|-------------|
| Key | Value | Description |

The **Body** tab is selected, showing the JSON response:

```
1 {
2   "nome": "Karina Almeida",
3   "email": "karina.almeida@fatec.br",
4   "perfil": "GERENTE",
5   "equipes": [
6     "Suporte",
7     "Desenvolvimento"
8   ]
9 }
```

The status bar at the bottom shows a **200 OK** response with a 201 ms duration and 326 B of data.

PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli

