

Desenvolvimento de APIs REST

06 - Relacionamento entre entidades

(continuação)

- Relacionamentos Um para Um

- Relacionamentos Um para Muitos / Muitos para Um
- Relacionamentos Muitos para Muitos
- Operações em Cascata



- H2 - Banco em Memória e Disco
- Relacionamento Embedded
- Enum e Exceções
- Herança

Relacionamento Um para Um



O relacionamento um para um, também conhecido como one-to-one, é utilizado para dividir uma tabela em duas, para deixar as informações de forma mais organizada.

Utilizando o projeto da **aula anterior** onde utilizamos o h2 como banco de dados.

inserir a classe **Proprietario** abaixo:

A linha abaixo é opcional para classe **Proprietario** geralmente mapeamos somente o lado onde vai ter a chave estrangeira do relacionamento.

```
@OneToOne(mappedBy = "proprietario")
private Veiculo veiculo;
```

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

    private String nome;

    //... gets e sets
    @Override
    public int hashCode() {
        return Objects.hash(id);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Proprietario other = (Proprietario) obj;
        return Objects.equals(id, other.id);
    }
}
```

- Inserir **Getter** e **Setter**
- **Equals** e **HashCode** para o atributo id

Relacionamento Um para Um



```
@Entity
@Table(name="veiculo")
public class Veiculo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message="Preencha a placa")
    @Size(max=7)
    @Column(nullable = false, length=7)
    private String placa;

    @NotBlank(message="Preencha a marca")
    @Size(max=30)
    @Column(nullable = false, length=30)
    private String marca;

    @NotBlank(message="Preencha o modelo")
    @Size(max=40)
    @Column(nullable = false, length=40)
    private String modelo;

    @Embedded @Valid
    private Caracteristica caracteristica;

    @OneToOne
    @JoinColumn(name="id_proprietario")
    private Proprietario proprietario;
```

Adicionar o atributo proprietário na classe **Veiculo**

Podemos mapear os dois lados do relacionamento, mas geralmente mapeamentos no lado que contém a chave estrangeira do relacionamento.

A propriedade **JoinColumn** é onde definimos qual será a chave estrangeira na tabela.

Relacionamento Um para Um



Vamos executar a aplicação, podemos verificar o resultado no console do STS e também no **h2**.

```
2022-09-04 13:10:59.754 INFO 1410504 --- [ restartedMain] o.s.b.d.h2.h2console.H2ConsoleConfiguration : h2 console available at /
2022-09-04 13:10:59.808 INFO 1410504 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing Pers
2022-09-04 13:10:59.812 INFO 1410504 --- [ restartedMain] org.hibernate.dialect.Dialect : HHH000400: Using dialect:
Hibernate: alter table veiculo add column id_proprietario bigint
Hibernate: alter table veiculo add constraint FK9odavigr8n5wtjkbuior6dil4 foreign key (id_proprietario) references proprietario
2022-09-04 13:11:00.075 INFO 1410504 --- [ restartedMain] o.h.e.t.j.p.i.StaPlatformInitiator : hnn000450: Using StaPlatfio
2022-09-04 13:11:00.079 INFO 1410504 --- [ restartedMain] i.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManager
```

Verificando as alterações no **h2**
<http://localhost:8080/h2-console/>

jdbc:h2:file:./h2/banco
FORNECEDOR
PROPRIETARIO
ID
NOME
Indexes
VEICULO
ID
ANO
CATEGORIA
CHASSI
COMBUSTIVEL
COR
RENAVAM
MARCA
MODELO
PLACA
ID_PROPRIETARIO
Indexes

Relacionamento Um para Um



Vamos criar a interface **ProprietarioRepository** e a classe **ProprietarioController**

```
@Repository
public interface ProprietarioRepository extends JpaRepository<Proprietario, Long>{

}
```

```
@RestController
public class ProprietarioController {

    @Autowired
    private ProprietarioRepository proprietarioRepository;

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Proprietario inserir(@RequestBody Proprietario proprietario){
        return proprietarioRepository.save(proprietario);
    }

    @PostMapping("/lista")
    @ResponseStatus(HttpStatus.CREATED)
    public List<Proprietario> inserirVarios(@RequestBody List<Proprietario> proprietarios) {
        return proprietarioRepository.saveAll(proprietarios);
    }

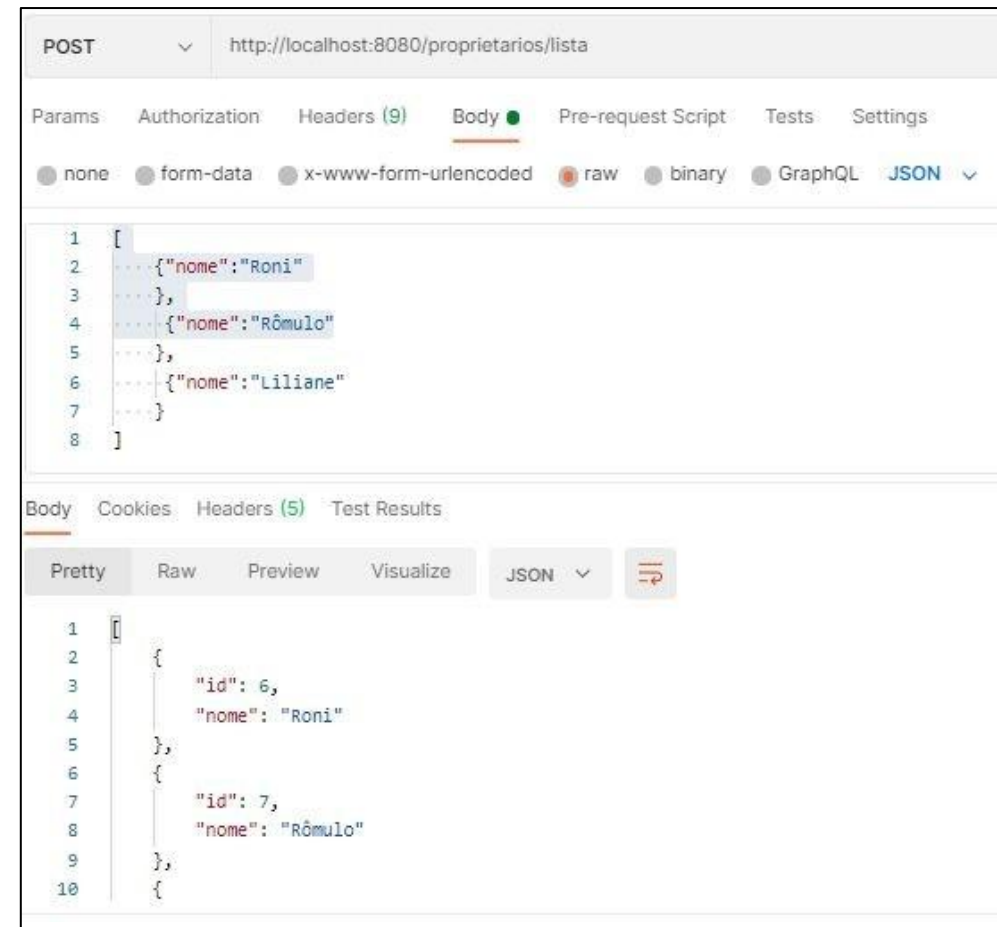
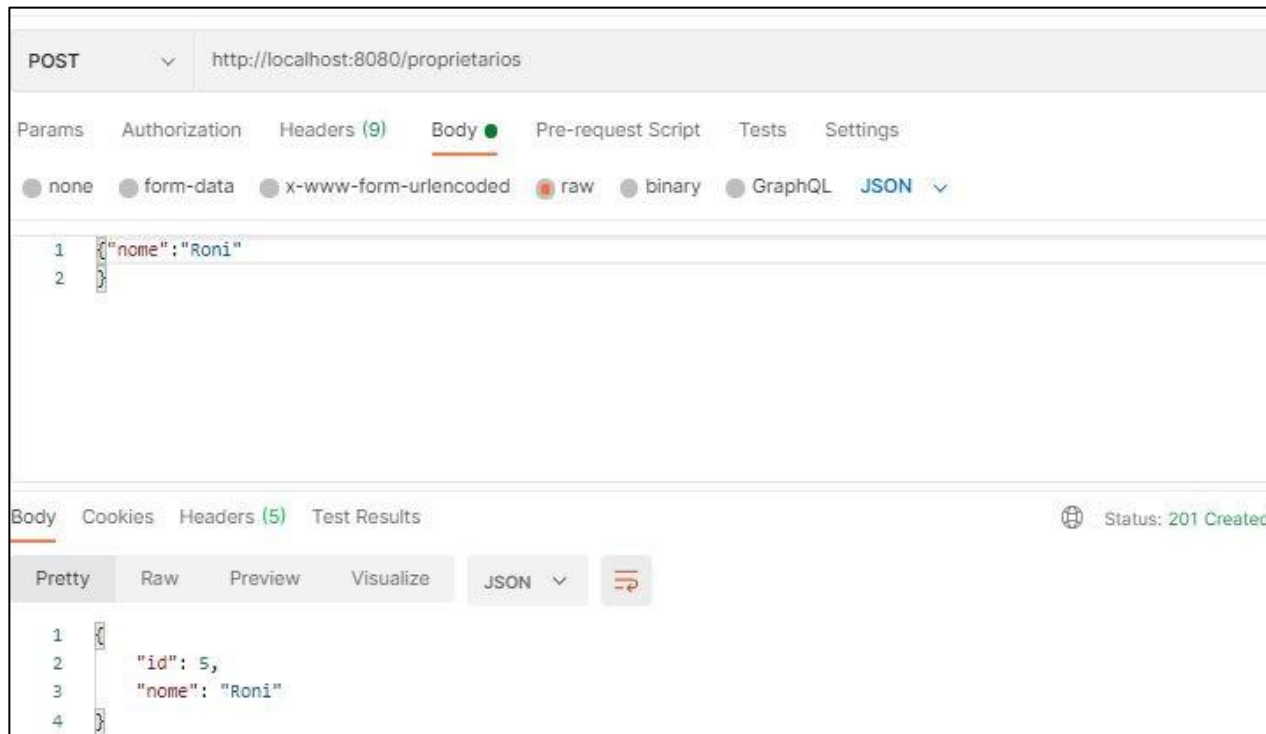
    @GetMapping
    public ResponseEntity<List<Proprietario>> listar(){
        List<Proprietario> proprietarios = proprietarioRepository.findAll();
        return ResponseEntity.ok(proprietarios);
    }

}
```

Relacionamento Um para Um



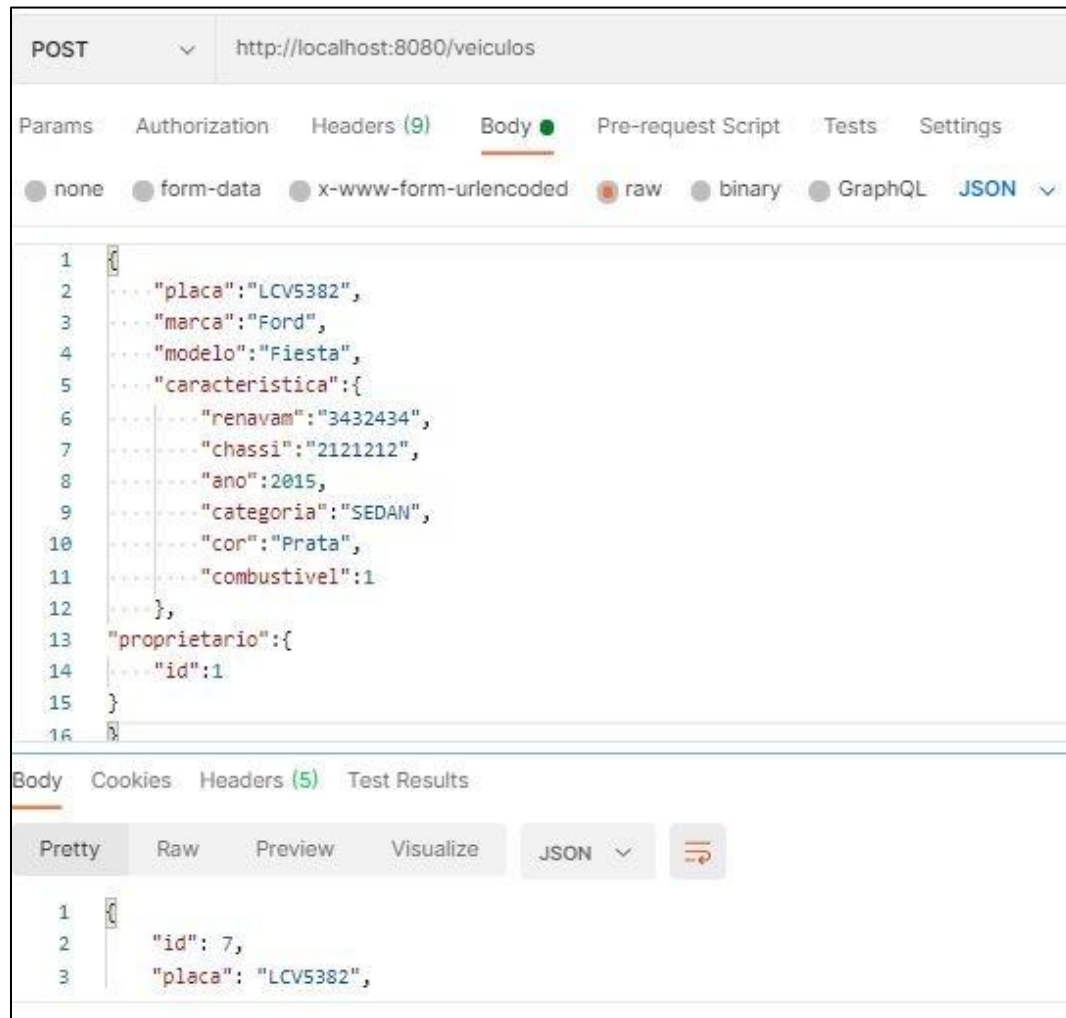
Testando no Postman as inclusões



Relacionamento Um para Um



Testando no Postman o cadastro do proprietário do veículo. Não é necessário preencher todos os campos somente o id do veículo.



Fetch.Lazy e Fetch.Eager



fetch.LAZY (buscar preguiçoso)

Os mapeamentos do Hibernate são Lazy por padrão. Quando acessamos o objeto ele não traz instantaneamente todas as dependências. Somente quando precisamos do atributo dependente é que a pesquisa é realizada, economizando memória.

fetch.EAGER (buscar ansioso)

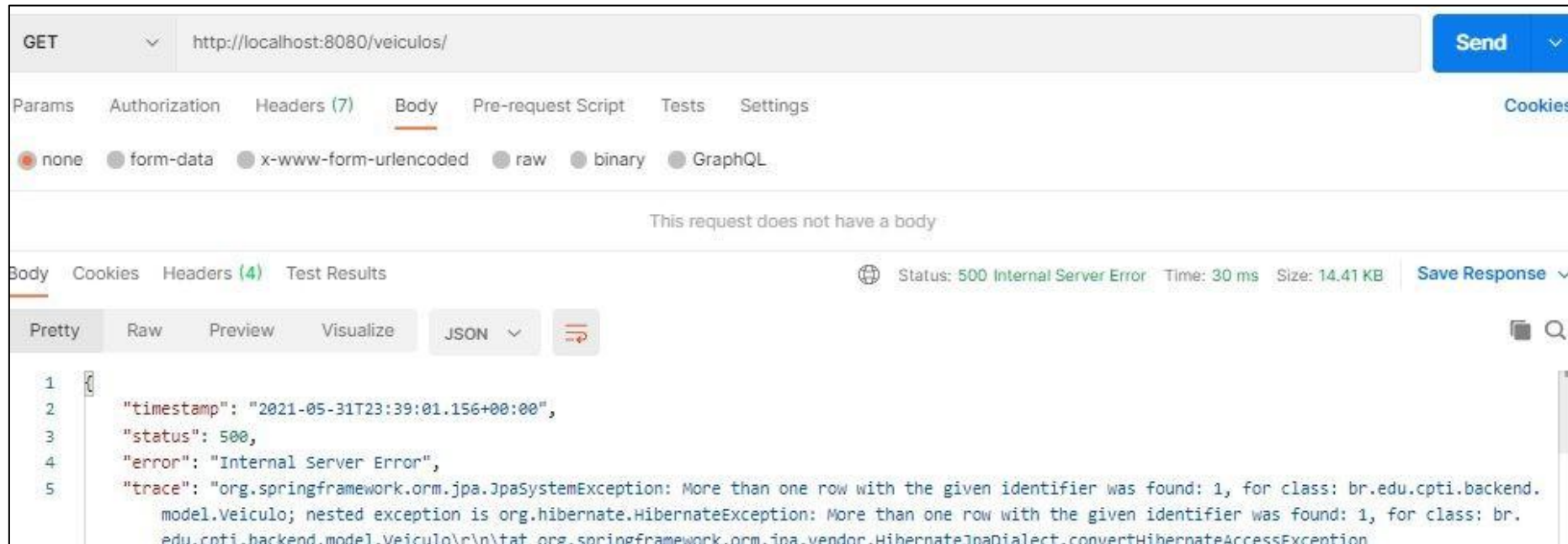
Retorna tudo que está dentro do objeto, se existir um relacionamento 1 para N, será carregado todas as referências dele.

Obs: Esta propriedade tem problemas com performance, não sendo quase utilizado.

Fetch.Lazy e Fetch.Eager



Ao listar todos veículos temos o erro abaixo:



Inserir na classe Veiculo:

```
@OneToOne(fetch = FetchType.EAGER)
@JoinColumn(name="id_proprietario")
private Proprietario proprietario;
```

Um para Muitos / Muitos para Um



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

    @Column(name="data_entrada")
    private LocalDate dataEntrada;

    @Column(name="data_saida")
    private LocalDate dataSaida;

    @Column
    private String obs;

    @ManyToOne
    @JoinColumn(name="id_veiculo")
    private Veiculo veiculo;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

Vamos inserir a classe **Manutencao**

Várias manutenções para um determinado veículo.

@JoinColumn – Estamos dizendo qual é a coluna da tabela que é a chave estrangeira.

Inserir **Getter** e **Setter**
Equals e **HashCode** para o atributo **id**

Um para Muitos / Muitos para Um



Vamos inserir o atributo **manutencoes** em destaque na classe **Veiculo**

Um veículo pode ter uma ou várias manutenções

mappedBy – Estamos dizendo qual é o nome do atributo da classe **Manutencao** que contém o relacionamento

```
@Entity
@Table(name="veiculo")
public class Veiculo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message="Preencha a placa")
    @Size(max=7)
    @Column(nullable = false, length=7)
    private String placa;

    @NotBlank(message="Preencha a marca")
    @Size(max=30)
    @Column(nullable = false, length=30)
    private String marca;

    @NotBlank(message="Preencha o modelo")
    @Size(max=40)
    @Column(nullable = false, length=40)
    private String modelo;

    @Embedded @Valid
    private Caracteristica caracteristica;

    @OneToOne(fetch = FetchType.EAGER)
    @JoinColumn(name="id_proprietario")
    private Proprietario proprietario;

    @OneToMany(mappedBy = "veiculo")
    private List<Manutencao> manutencoes;
```

Um para Muitos / Muitos para Um



Executando a aplicação verificamos no console que o hibernate criou a tabela **manutencao** no banco de dados

```
2022-09-04 14:59:32.929 INFO 1410504 --- [ restartedMain] org.hibernate.dialect.Dialect : h2000400: Using dialect: org.hibernate.dialect.H2Dialect
Hibernate: create table manutencao (id bigint generated by default as identity, data_entrada date, data_saida date, obs varchar(255), id_veiculo bigint, primary key (id))
Hibernate: alter table manutencao add constraint FKdfe27xk4hngxvnrfvnsh6a0d foreign key (id_veiculo) references veiculo
2022-09-04 14:59:33.008 INFO 1410504 --- [ restartedMain] o.h.e.t.j.p.i.itaPlatformInitiator : HHH000400: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-09-04 14:59:33.008 INFO 1410504 --- [ restartedMain] i.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
```

Verificando no **h2** a tabela foi criada com a chave estrangeira da tabela **veiculo**

jdbc:h2:file:./h2/banco
FORNECEDOR
MANUTENCAO
ID
DATA_ENTRADA
DATA_SAIDA
OBS
ID_VEICULO
Indexes
PROPRIETARIO
VEICULO
ID
ANO
CATEGORIA
CHASSI
COMBUSTIVEL
COR
RENAVAM
MARCA
MODELO
PLACA
ID_PROPRIETARIO
Indexes

Um para Muitos / Muitos para Um



```
@RestController
@RequestMapping("/manutencoes")
public class ManutencaoController {
    @Autowired
    private ManutencaoRepository manutencaoRepository;
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Manutencao inserir(@RequestBody Manutencao manutencao) {
        return manutencaoRepository.save(manutencao);
    }
    @GetMapping
    public ResponseEntity<List<Manutencao>> listar(){
        List<Manutencao> manutencoes = manutencaoRepository.findAll();
        return ResponseEntity.ok(manutencoes);
    }
    @GetMapping("/{id}")
    public ResponseEntity<Manutencao> buscar(@PathVariable Long id) {
        Optional<Manutencao> manutencao = manutencaoRepository.findById(id);
        if (manutencao.isPresent()){
            return ResponseEntity.ok(manutencao.get());
        }
        return ResponseEntity.notFound().build();
    }
}
```

Inserir o **controller** e o **repository** para manutencao

```
@Repository
public interface ManutencaoRepository extends
    JpaRepository<Manutencao, Long>{

}
```

Um para Muitos / Muitos para Um



Testando no Postman

POST http://localhost:8080/manutencoes

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "dataEntrada": "2021-04-20",
3   "dataSaida": "2021-05-10",
4   "obs": "Troca de Óleo",
5   "veiculo": {
6     "id": 1
7   }
8 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 3,
3   "dataEntrada": "2021-04-20",
4   "dataSaida": "2021-05-10",
5   "obs": "Troca de Óleo"
6 }
```

Status: 201 Created

GET http://localhost:8080/manutencoes

Params Authorization Headers (7) Body Pre-request Script Tests

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "dataEntrada": "2021-05-21",
4   "dataSaida": "2021-05-22",
5   "obs": "Veículo OK"
6 },
7 {
8   "id": 2,
9   "dataEntrada": "2021-05-23",
10  "dataSaida": "2021-05-23",
11  "obs": "Veículo Revisado"
12 },
13 {
14   "id": 3,
15   "dataEntrada": "2021-04-20",
16   "dataSaida": "2021-05-10",
17   "obs": "Troca de Óleo"
18 }
19 ]
20 }
```

Um para Muitos / Muitos para Um



Testando no Postman

GET ▼ http://localhost:8080/manutencoes/2

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "id": 2,
3   "dataEntrada": "2021-05-23",
4   "dataSaida": "2021-05-23",
5   "obs": "Veículo Revisado"
6 }
```

GET ▼ http://localhost:8080/veiculos/8

Params Authorization Headers (7) Body Pre-request Script

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼

```
14 {
15   "proprietario": {
16     "id": 1,
17     "nome": "Roni"
18   },
19   "manutencoes": [
20     {
21       "id": 1,
22       "dataEntrada": "2021-05-21",
23       "dataSaida": "2021-05-22",
24       "obs": "Veículo OK"
25     },
26     {
27       "id": 2,
28       "dataEntrada": "2021-05-23",
29       "dataSaida": "2021-05-23",
30       "obs": "Veículo Revisado"
31     }
32   ]
33 }
```


Muitos para Muitos



Vamos criar a classe **Servico** para utilizar neste exemplo

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

    @Column
    private String descricao;

    @Column
    private BigDecimal valor;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

Inserir **Getter e Setter**
Equals e hashCode para o atributo **id**

Muitos para Muitos



Vamos inserir o atributo **serviços** na classe **Manutencao**

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

    @Column(name="data_entrada")
    private LocalDate dataEntrada;

    @Column(name="data_saida")
    private LocalDate dataSaida;

    @Column
    private String obs;

    @ManyToOne
    @JoinColumn(name="id_veiculo")
    private Veiculo veiculo;

    @ManyToMany
    @JoinTable(name="manutencao_servico",
        joinColumns = @JoinColumn(name="id_manutencao"),
        inverseJoinColumns = @JoinColumn(name="id_servico"))
    private List<Servico> servicos;
```

Um veículo pode ter mais de um tipo de serviço para fazer no dia da manutenção como por exemplo, troca de óleo, revisão e outros.

Um serviço pode estar em diversas manutenções de um determinado veículo.

Uma manutenção de um veículo pode ter diversos serviços.

Quando temos um relacionamento muitos para muitos é criada uma tabela intermediária para este modelo.

@JoinTable - serve para configurar a tabela intermediária. Precisamos ter duas colunas uma para referência ao tipo de serviço (id_servico) e outra para o identificador da manutenção (id_manutencao).

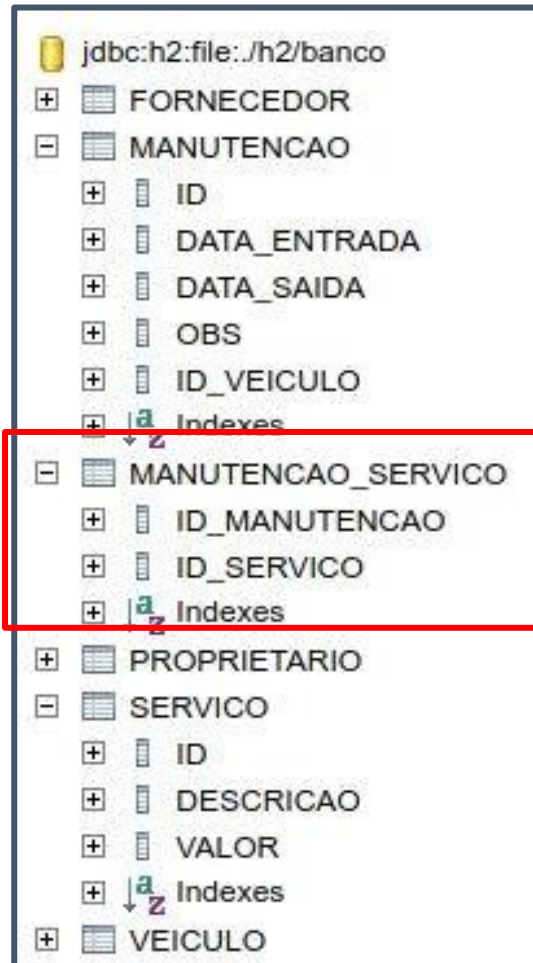
joinColumns - vamos configurar as colunas

@JoinColumn(name = "id_manutencao") - coluna referente a tabela **manutencao**

@inverseJoinColumns - responsável por mapear colunas do lado inverso do relacionamento.

@JoinColumn(name = "id_servico") - coluna referente a tabela **servico**

Muitos para Muitos



Vamos executar a aplicação e verificar no **h2**.

Cascade



```
@ManyToOne(cascade = CascadeType.ALL)
@ManyToMany(cascade = CascadeType.ALL)
@OneToOne(cascade = CascadeType.ALL)
```

- ALL - propaga todas as operações do “pai” para os “filhos”. Ex: efetuar uma operação e uma manutenção, todos os serviços associados sofreram a mesma operação (persistência, exclusão, etc)
- PERSIST - propaga as operações de persistência (insert)
- MERGE - operações de atualização (insert para “filhos” novos e update nos que foram alterados)
- REMOVE - ao remover o “pai”, os “filhos” também serão removidos
- DETACH - remove a entidades do “contexto” de persistência
- LOCK - “re-adiciona” a entidade ao “contexto” de persistência
- REFRESH - atualiza a instância com os dados do banco de dados
- REPLICATE - apenas quando existe mais de um banco de dados (replicação) e se deseja manter os dados sincronizados
- SAVE_UPDATE - propaga operações de inserção e update

Mais informações <https://www.baeldung.com/jpa-cascade-types>

Exercícios



Exercícios

Vamos acessar o projeto em que foi utilizado o banco de dados no **Postgres** para fazer este exercício.
Inserir o script abaixo no banco.

```
CREATE TABLE pedido(id_pedido serial PRIMARY KEY, data_pedido date, hora_pedido time, data_entrega date, id_cliente bigint,  
FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente));  
  
INSERT INTO PEDIDO (data_pedido, hora_pedido, data_entrega, id_cliente) VALUES ('2021-10-20', '20:00', '2021-10-28',1);  
INSERT INTO PEDIDO (data_pedido, hora_pedido, data_entrega, id_cliente) VALUES ('2021-09-21', '21:00', '2021-09-26',2);  
INSERT INTO PEDIDO (data_pedido, hora_pedido, data_entrega, id_cliente) VALUES ('2021-10-19', '14:00', '2021-10-21',1);  
INSERT INTO PEDIDO (data_pedido, hora_pedido, data_entrega, id_cliente) VALUES ('2021-10-16', '12:00', '2021-10-18',2);
```

Exercício One to Many / Many To One



```
@Entity
public class Pedido {
    @Id

    @GeneratedValue(strategy=Generation
Type.IDENTITY)
    @Column(name="id_pedido")
    private Long id;

    @Column(name="data_pedido")
    private LocalDate dataPedido;

    @Column(name="hora_pedido")
    private LocalTime horaPedido;

    @Column(name="data_entrega")
    private LocalDate dataEntrega;

    public Long getId() {
        return id;
    }
}
```

Vamos inserir a classe **Pedido**

Fazemos o mapeamento em que temos vários pedidos de um determinado cliente.

@JoinColumn – Estamos dizendo qual é a coluna da tabela que é a chave estrangeira.

Inserir **Getter** e **Setter**

Equals e **HashCode** para o atributo **id**

Exercício One to Many / Many To One



Vamos inserir a interface **PedidoRepository**

```
@Repository
public interface PedidoRepository extends
    JpaRepository<Pedido, Long> {
}
```

Vamos inserir a classe **PedidoController** para listar todos pedidos e por id

```
@RestController
@RequestMapping("/pedidos")
public class PedidoController {
    @Autowired
    private PedidoRepository pedidoRepository;
    @GetMapping
    public List<Pedido> listar() {
        return pedidoRepository.findAll();
    }
    @GetMapping("/{id}")
    public ResponseEntity<Pedido> buscar(@PathVariable Long id) {
        Optional<Pedido> pedido = pedidoRepository.findById(id);
        if (pedido.isPresent()) {
            return ResponseEntity.ok(pedido.get());
        }
        return ResponseEntity.notFound().build();
    }
}
```

Exercício One to Many / Many To One



Vamos testar no Postman

The screenshot shows the Postman interface for a GET request to `http://localhost:8080/pedidos/`. The response status is 200 OK, with a time of 16 ms and a size of 1.33 KB. The response body is displayed in JSON format, showing a single object with the following structure:

```
1 {
2   "id": 1,
3   "dataPedido": "2021-03-15",
4   "horaPedido": "20:00:00",
5   "dataEntrega": "2021-03-16",
6   "cliente": {
7     "id": 1,
8     "nome": "Joaquim",
9     "cpf": "49422854024",
10    "email": "joaquim@gmail.com",
11    "endereco": {
12      "logradouro": "Rua Fonseca Ramos",
13      "numero": "134",
14      "bairro": "Centro",
15      "cidade": "Juiz de Fora",
16    }
17  }
18 }
```


Exercício One to Many / Many To One



Adicionar o método para inserir o pedido

Testar no Postman

```
@PostMapping
@ResponseStatus(HttpStatus.CREATED)
public Pedido inserir(@Valid @RequestBody Pedido pedido) {
    return pedidoRepository.save(pedido);
}
```

POST http://localhost:8080/pedidos

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "dataPedido": "2021-03-21",
3   "horaPedido": "21:02:02",
4   "dataEntrega": null,
5   "cliente": {
6     "id": 2
7   }
8 }
```

Body Cookies Headers (5) Test Results 201 Created 143 ms 317 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 5,
3   "dataPedido": "2021-03-21",
4   "horaPedido": "21:02:02",
5   "dataEntrega": null,
6   "cliente": {
7     "id": 2,
8     "nome": null,
9     "cpf": null,
10    "email": null,
11    "endereco": null
12  }
13 }
```

GET http://localhost:8080/pedidos/1

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies Beautify

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results 200 OK 11 ms 446 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "dataPedido": "2021-10-20",
4   "horaPedido": "20:00:00",
5   "dataEntrega": "2021-10-28",
6   "cliente": {
7     "id": 1,
8     "nome": "Joaquina",
9     "cpf": "943536",
10    "email": "joaquim@gmail.com",
11    "endereco": {
12      "logradouro": "Rua Fonseca Ramos",

```

Exercício One to One



O relacionamento um para um, também conhecido como one-to-one, é utilizado para dividir uma tabela em duas, para deixar as informações de forma mais organizada.



Um cliente possui somente um **documento**.

Um documento pertence a um determinado cliente.

Adicionar o script abaixo no banco de dados do **Postgres**

```
CREATE SEQUENCE documento_id_seq;
CREATE TABLE documento (
  id_documento bigint PRIMARY KEY DEFAULT nextval('documento_id_seq'),
  identidade varchar(15),
  orgao_expositor varchar(20),
  cnh varchar(20),
  titulo_eleitor varchar(15),
  id_cliente bigint unique,
  FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente)
);

INSERT INTO documento (identidade, orgao_expositor, cnh, titulo_eleitor, id_cliente)
VALUES ('098442', 'IFP', '234343242', '234243', 1);
```

Exercício One to One



Criar a classe **Documento**

```
@Entity
public class Documento {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id_documento")
    private Long id;

    @Column
    private String identidade;

    @Column(name="orgao_expositor")
    private String orgaoExpositor;

    @Column
    private String cnh;

    @Column(name="titulo_eleitor")
    private String tituloEleitor;

    @OneToOne
    @JoinColumn(name="id_cliente")
    private Cliente cliente;
```

Podemos mapear nos dois lados do relacionamento mas geralmente mapeamentos no lado que contém a chave estrangeira do relacionamento.

A propriedade **JoinColumn** é onde definimos qual será a chave estrangeira na tabela

Inserir **Getter** e **Setter**

Equals e **HashCode** para o atributo **id**