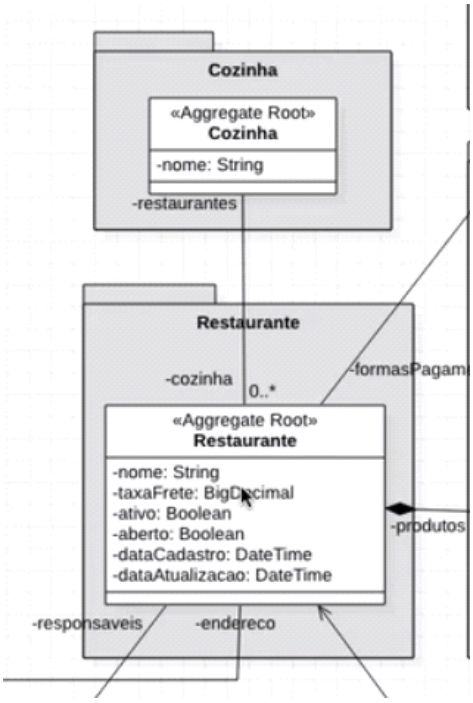
6.1. Mapeando relacionamento bidirecional com @OneToMany

sábado, 18 de fevereiro de 2023 11:27



Vamos fazer o mapeamento bidirecional, ou seja, a entidade Restaurante tem conhecimento (serializa/vincula) de Cozinha, e vice e versa, o relacionamento pode ser @OneToOne ou @ManyToOne e podemos anotar se a entidade ter como propriedade a outra entidade. Se for um relacionamento unidirecional, uma entidade não tem conhecimento da outra, mas a outra tem conhecimento.

Restaurante.java

```
20      @Column(nullable = false)
21      private String nome;
22
23      @Column(name = "taxa_frete", nullable = false)
24      private BigDecimal taxaFrete;
25
26      @JsonIgnore
27      @ManyToOne
28      @JoinColumn(name = "cozinha_id", nullable = false)
29      private Cozinha cozinha;
30
31  }
```

A entidade Restaurante tem conhecimento da entidade Cozinha e está anotada com @ManyToOne. Dica: a entidade que tem anotada em uma de suas propriedades com Many primeiro, a mesma entidade possui muitas entidades da propriedade (many) ex: Muitos restaurantes tem 1 (uma) cozinha.

Agora na entidade que está no sentido oposto do mapeamento é anotada com @OneToMany, como é Um Para Muitos, o Muitos simboliza uma lista da entidade e contém a propriedade na anotação dizendo qual propriedade na outra classe que a mesma tá vinculada, no caso está mapeada por cozinha. Dica: se no final do mapeamento estiver escrito ToMany deve-se ter a propriedade mappedBy.

A anotação @JsonIgnore é necessário, pois sem ela, a serialização acontece em todas as entidaes, e caso uma entidade conter uma entidade como propriedade, o Jackson serializa também, e isso se torna uma serialização circular, um loop, portanto, é necessário Ignorar a serialização em uma das entidadaes

6.2. Mapeando relacionamento muitos-para-muitos com @ManyToMany

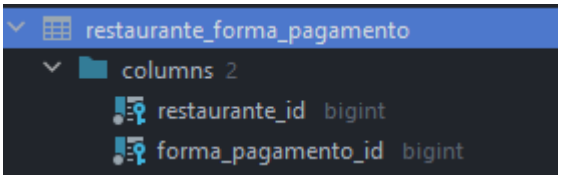
sábado, 18 de fevereiro de 2023 14:32

```
@ManyToMany
@JoinTable(name = "restaurante_forma_pagamento",
           joinColumns = @ JoinColumn(name = "restaurante_id"),
           inverseJoinColumns = @JoinColumn(name = "forma_pagamento_id"))
private List<FormaPagamento> formasPagamento = new ArrayList<>();
```

@JoinTable : tem 3 propriedades, name, joinColumns e inverseJoinColumns

joinColumns: recebe uma anotação @JoinColumn(name) como nome da tabela que representa a entidade na qual está anotada

inverseJoinColumns = recebe uma anotação @JoinColumn(name) como nome da tabela que representa a outra tabela da tabela de associação

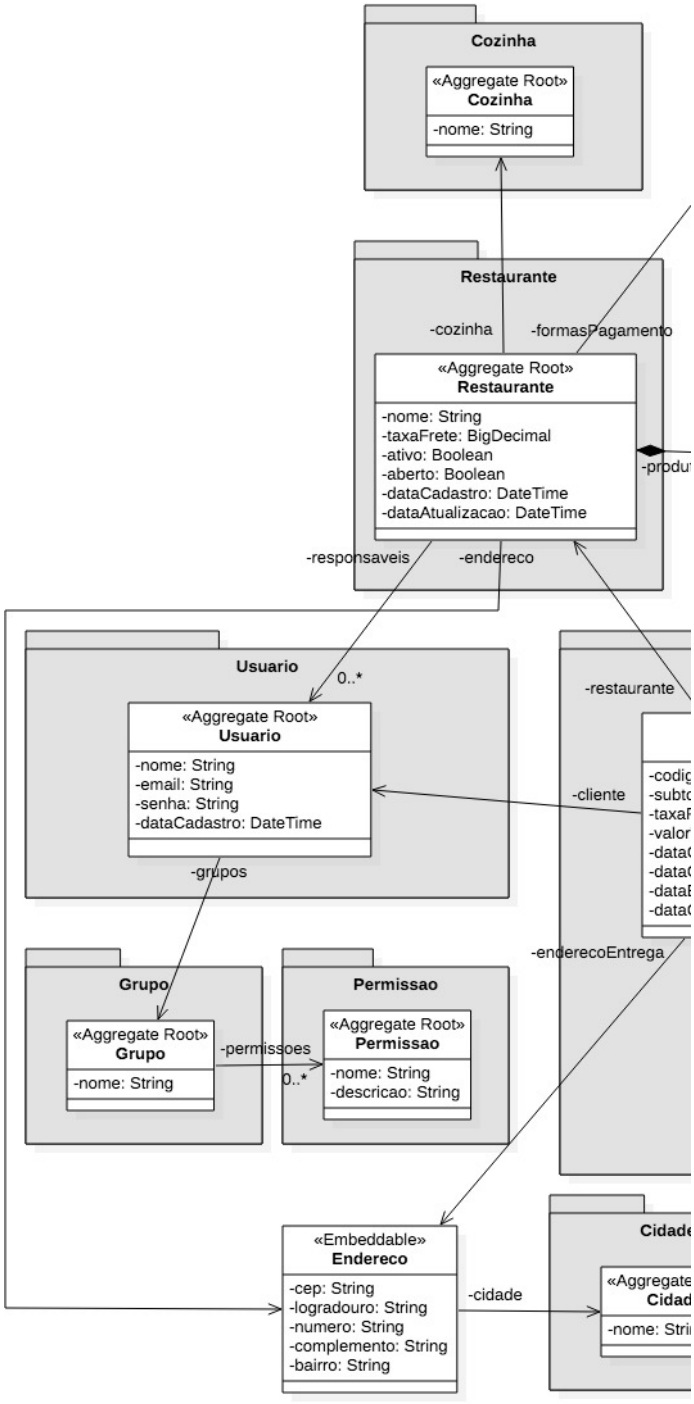


6.3. Analisando o impacto do relacionamento muitos-para-muitos na REST API

sábado, 18 de fevereiro de 2023 15:16

6.4. Mapeando classes incorporáveis com @Embedded e @Embeddable

sábado, 18 de fevereiro de 2023 15:59



No nosso diagrama de classes no modelo do domínio, a entidade Restaurante tem um endereco associado, mas não está na mesma classe, componentizamos a classe.
Objetos embutidos são componentes de uma entidade que não está na mesma classe mas que refletem na tabela da entidade

Separamos em classes diferentes mas pertencentes de uma mesma tabela, esses

são objetos embutidos.

```java
1 usage
@Data
@Embeddable//indica que a classe é incorporada, parte de uma entidade
public class Endereco {

    @Column(name="endereco_cep")
    private String cep;

    @Column(name="endereco_logradouro")
    private String logradouro;

    @Column(name="endereco__numero")
    private String numero;

    @Column(name="endereco_complemento")
    private String complemento;

    @Column(name="endereco_bairro")
    private String bairro;

    @ManyToOne
    @JoinColumn(name = "endereco_cidade_id")
    private Cidade cidade;
}
```

Primeiro criamos a classe que será embutida e anotamos com @Embeddable

```java
@Embedded
private Endereco endereco;
```

Na classe criamos a propriedade da classe que é "Embutível" (@Embeddable) e anotamos com @Embedded = Embutida.

6.5. Testando e analisando o impacto da incorporação de classe na REST API

sábado, 18 de fevereiro de 2023 19:11

6.6. Mapeando propriedades com @CreationTimestamp e @UpdateTimestamp

sábado, 18 de fevereiro de 2023 22:11

```java
@CreationTimestamp
@Column(nullable = false)
private LocalDateTime dataCadastro;

@UpdateTimestamp
@Column(nullable = false)
private LocalDateTime dataAtualizacao;
```

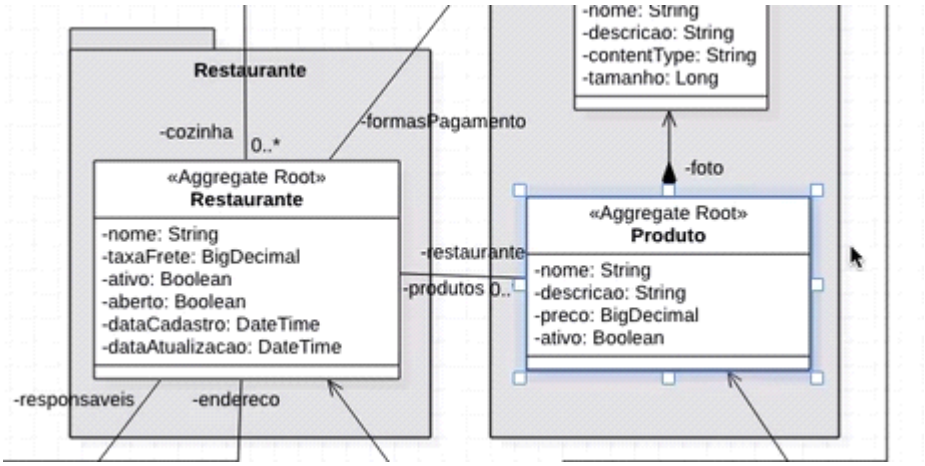
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	data_atualizacao	data_cadastro	endereco_bairro	endereco_cep	endereco_complemento	endereco_
1	2023-02-19 01:15:05	2023-02-19 01:15:05	Centro	38400-999	NULL	Rua João P
2	2023-02-19 01:15:05	2023-02-19 01:15:05	NULL	NULL	NULL	NULL
3	2023-02-19 01:15:05	2023-02-19 01:15:05	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```sql
insert into restaurante
(id, nome, taxa_frete, cozinha_id, data_cadastro, data_atualizacao, endereco_cidade_id,
endereco_cep, endereco_logradouro, endereco_numero, endereco_bairro)
values
(1, 'Thai Gourmet', 10, 1, utc_timestamp, utc_timestamp, 1,
'38400-999', 'Rua João Pinheiro', '1000', 'Centro');

insert into restaurante
(id, nome, taxa_frete, cozinha_id, data_cadastro, data_atualizacao)
values
 (2, 'Thai Delivery', 9.50, 1, utc_timestamp, utc_timestamp);

insert into restaurante
(id, nome, taxa_frete, cozinha_id, data_cadastro, data_atualizacao)
values
(3, 'Tuk Tuk Comida Indiana', 15.00, 2, utc_timestamp, utc_timestamp);
```

6.7. Desafio mapeando relacionamento muitos-para-um

domingo, 19 de fevereiro de 2023 14:18



Somente a direção Produto para Restaurante @ManyToOne muitos produtos possuem um restaurante

```java
@Data
@Entity
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
public class Produto {

    @EqualsAndHashCode.Include
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable=false)
    private String nome;

    private String descricao;

    @Column(nullable=false)
    private BigDecimal preco;

    @Column(nullable=false)
    private Boolean ativo;

    @ManyToOne
    private Restaurante restaurante;
}
```

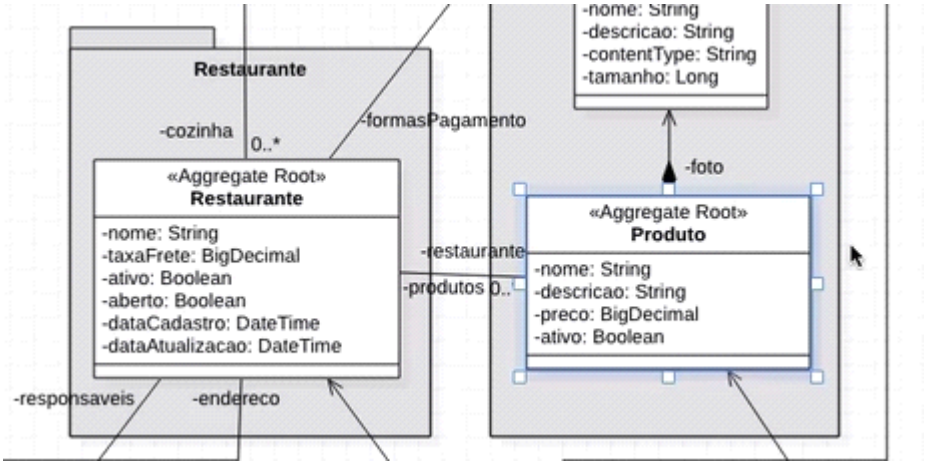
@Data -> Setter e Getter, EqualsAndHashCode
@EqualAndHashCode -> onlyExplicitlyIncluded = true

id	ativo	descricao	nome	preco	restaurante_id
1	1	Macaxeira Cozida para Café	Macaxeira	20.00	1
2	1	Deliciosa carne suína ao molho especial	Porco com molho agridoce	78.90	1
3	1	16 camarões grandes ao molho picante	Camarão tailandês	110.00	1
4	1	Salada de folhas com cortes finos de carne bovi...	Salada picante com carne grelhada	87.20	2
5	1	Pão tradicional indiano com cobertura de alho	Garlic Naan	21.00	3
6	1	Cubos de frango preparados com molho curry e...	Murg Curry	43.00	3
7	1	Corte macio e suculento, com dois dedos de esp...	Bife Ancho	79.00	4
8	1	Corte muito saboroso, com um osso em formato...	T-Bone	89.00	4
9	1	Sandubão com muito queijo, hamburger bovino,...	Sanduíche X-Tudo	19.00	5
10	1	Acompanha farinha, mandioca e vinagrete	Espetinho de Cupim	8.00	6
NULL	NULL	NULL	NULL	NULL	NULL

6.8. Desafio mapeando relacionamento um-para-muitos

domingo, 19 de fevereiro de 2023 14:31

Mapeando de Restaurante Para Produto



Um Restaurante Possui Muitos Produtos `@OneToMany` e como o sufixo é `ToMany` se deve colocar o `mappedBy`. É necessário que seja uma coleção de entidades, logo, uma lista, é preferível que seja inicializada na definição.

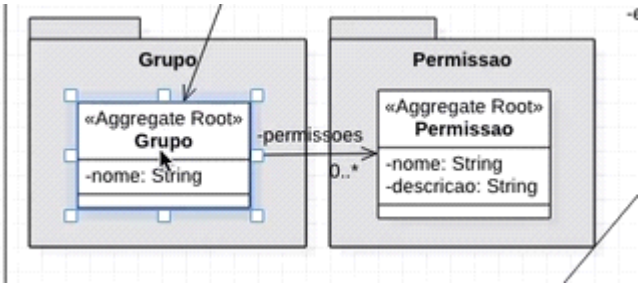
```
@JsonIgnore
@OneToMany(mappedBy = "restaurante")
private List<Produto> produtos = new ArrayList<>();
```

A anotação com sufixo ToMany e a coleção de entidade associada à ela não reflete no banco de dados, ou seja, a tabela Restaurante não possui um coluna contendo a lista de produtos. É criada somente a tabela Produto com as propriedades e o id de cada restaurante

id	ativo	descricao	nome	preco	restaurante_id
1	1	Macaxeira Cozida para Café	Macaxeira	20.00	1
2	1	Deliciosa carne suína ao molho especial	Porco com molho agridoce	78.90	1
3	1	16 camarões grandes ao molho picante	Camarão tailandês	110.00	1
4	1	Salada de folhas com cortes finos de carne bovi...	Salada picante com carne grelhada	87.20	2
5	1	Pão tradicional indiano com cobertura de alho	Garlic Naan	21.00	3
6	1	Cubos de frango preparados com molho curry e...	Murg Curry	43.00	3
7	1	Corte macio e suculento, com dois dedos de esp...	Bife Ancho	79.00	4
8	1	Corte muito saboroso, com um osso em formato...	T-Bone	89.00	4
9	1	Sandubão com muito queijo, hamburger bovino,...	Sanduíche X-Tudo	19.00	5
10	1	Acompanha farinha, mandioca e vinagrete	Espetinho de Cupim	8.00	6
NULL	NULL	NULL	NULL	NULL	NULL

6.9. Desafio mapeando relacionamentos muitos-para-muitos

domingo, 19 de fevereiro de 2023 17:54

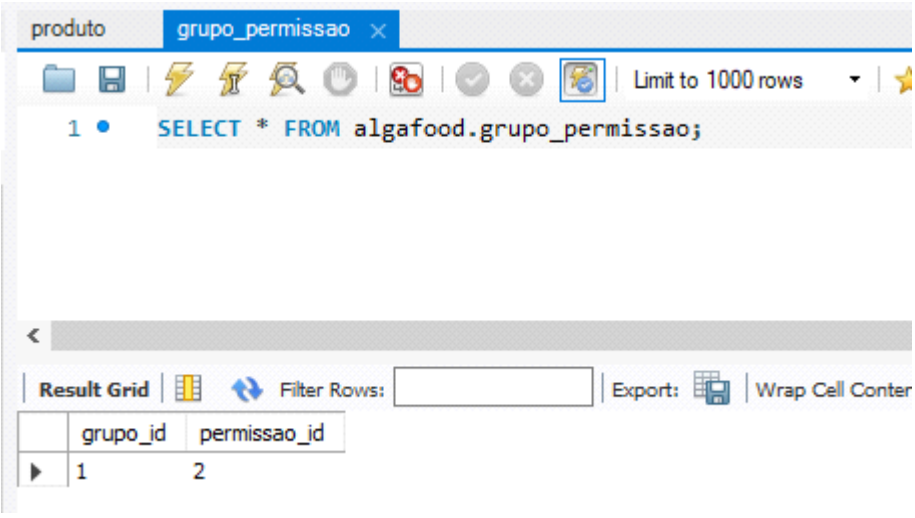


Criar a entidade Grupo, mapear e definir as associações Um Grupo Possui Muitas Permissões.

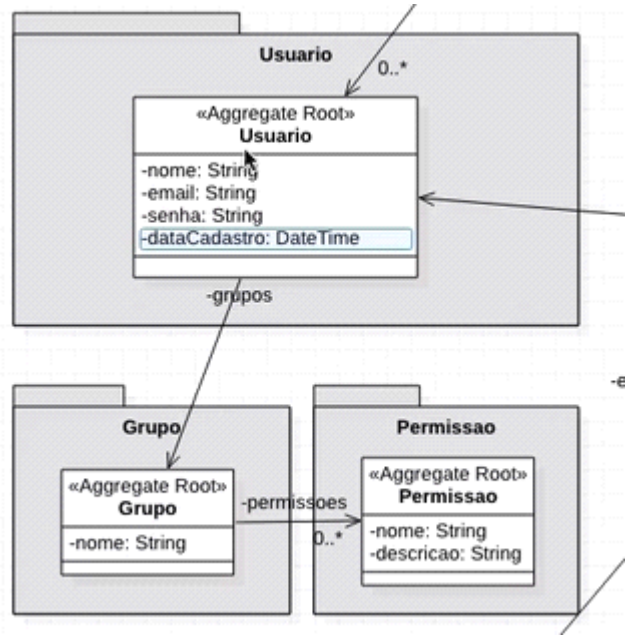
De acordo com o diagrama UML, a associação é unidirecional, a entidade Permissao não precisa conhecer a entidade Grupo, mas ela pode conter um relacionamento Muitos Para Muitos @ManyToMany. Muitos grupo possui muitas permissões e Muitas permissões são contidas por Muitos grupos.

```
@Entity
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
public class Grupo {

    @EqualsAndHashCode.Include
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String nome;

    @ManyToMany
    @JoinTable(name = "grupo_permissao",
            joinColumns = @JoinColumn(name = "grupo_id"),
            inverseJoinColumns = @JoinColumn(name = "permissao_id"))
    private List<Permissao> permissoes;
}
```

```java
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Entity
public class Permissao {

    @Id
    @EqualsAndHashCode.Include
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String nome;

    @Column(nullable = false)
    private String descricao;
}
```

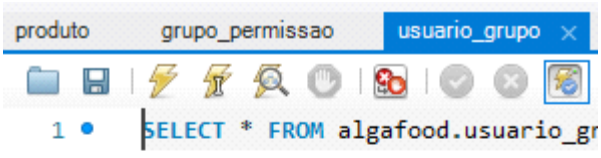


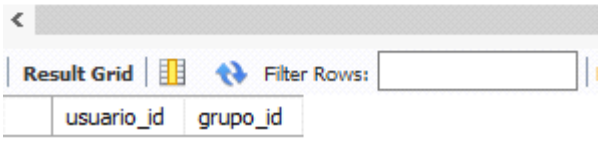
Agora a criação da entidade Usuario e o seu relacionamento com Grupo



A entidade usuário possui muitos grupos e muitos grupos possuem muitos usuários

```java
@Entity
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
public class Usuario {

    @Id
    @EqualsAndHashCode.Include
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String nome;

    @Column(nullable = false)
    private String senha;

    @Column(nullable = false)
    private String email;

    @CreationTimestamp
    @Column(columnDefinition = "datetime", nullable = false)
    private LocalDateTime dataCadastro;

    @ManyToMany
    @JoinTable(name = "usuario_grupo",
            joinColumns = @JoinColumn(name = "usuario_id"),
            inverseJoinColumns = @JoinColumn(name = "grupo_id"))
    private List<Grupo> grupos = new ArrayList<>();
}
```





6.10. Entendendo o Eager Loading

domingo, 19 de fevereiro de 2023 21:21

Classe restaurante:

```java
@EqualsAndHashCode.Include
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY) //provedor de persister
private Long id;

@Column(nullable = false)
private String nome;

@Column(name = "taxa_frete", nullable = false)
private BigDecimal taxaFrete;

//@JsonIgnore
@ManyToOne
@JoinColumn(name = "cozinha_id", nullable = false)
private Cozinha cozinha;

@CreationTimestamp
@Column(nullable = false, columnDefinition = "datetime")
private LocalDateTime dataCadastro;

@UpdateTimestamp
@Column(nullable = false, columnDefinition = "datetime")
private LocalDateTime dataAtualizacao;

@JsonIgnore
@ManyToMany
@JoinTable(name = "restaurante_forma_pagamento",
           joinColumns = @JoinColumn(name = "restaurante_id"),
           inverseJoinColumns = @JoinColumn(name = "forma_pagamento_id"))
private List<FormaPagamento> formasPagamento = new ArrayList<>();

@JsonIgnore
@Embedded
private Endereco endereco;
```

Na classe temos um mapeamento @ManyToOne e um objetivo embutido endereco, que contém uma cidade, logo, um objeto restaurante contém uma cozinha e uma cidade, e a cidade contém um estado.

Tabela restaurante contém 6 campos:

id	nome	cozinha_id	endereco_cidade_id
1	Thai Gourmet	1	1
2	Thai Delivery	1	NULL
3	Tuk Tuk Comida Indiana	2	NULL
4	Java Steakhouse	3	NULL
5	Lanchonete do Tio Sam	4	NULL
6	Bar da Maria	4	NULL
NULL	NULL	NULL	NULL

Cada campo traz consigo um id para cozinha e um id para cidade e cidade traz um id para estado.

Requisição para trazer todas os restaurantes:

```
2023-02-19 21:22:54.849  INFO 2684 --- [nio-8080-exec-2] o.s.web.s
Hibernate: select restaurant0_.id as id1_8_, restaurant0_.cozinha_
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome
Hibernate: select cidade0_.id as id1_0_0_, cidade0_.estado_id as e
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome
```

o 1º select de fato trás todos os 6 restaurantes da tabela restaurante
o 2º select traz a 1ª cozinha associada ao 1º restaurante da lista (6), existem 4 cozinhas cadastradas para 6 restaurantes.
o 3º select traz uma cidade associada a 1ª cozinha do 1º restaurante da lista e logo depois 1º estado associada a essa cidade.
o 4º, 5º e 6º select traz as cozinhas associadas aos outros 5 restaurantes da lista(6).

Foram tragas as 4 cozinhas em 4 selects para os 6 restaurantes, os objetos ficam cacheados para reutilização em outros objetos, por isso apenas 4 selects para 6 objetos, pois um desses objetos utilizavam a mesma cozinha.

O mesmo é válido para cidade, pois os 6 restaurantes estão associados a apenas 1 cidade, por isso somente 1 select para todos os 6 restaurantes.

A estratégia Eager Loading basicamente diz para carregar junto as entidades associadas a outras entidades que estão associadas, mesmo que na representação json não apresente tais entidades, pois o efeito é refletido na estratégia de recuperação dos dados a partir do mapeamento das entidades, não da representação. Não importa se for no mesmo select ou em vários selects, como no 3º select da requisição acima, quando no mesmo select foram trazidos a cidade e estado, não em selects diferentes, o Hibernate decide a melhor forma de trazer os dados.

6.11. Entendendo o Lazy Loading

segunda-feira, 20 de fevereiro de 2023 10:04

Carregamento preguiçoso - Só faz o carregamento da entidade somente se necessário. Toda associação com prefixo ToMany utiliza o carregamento Lazy. Exemplo:
Um restaurante possui uma associação ToMany com formasDePagamento, se precisar utilizar as formas de pagamento na representação do recurso ou dentro do código, o Hibernate faz o select de todas as associações

6.12. Alterando a estratégia de fetching para Lazy Loading

segunda-feira, 20 de fevereiro de 2023 10:32

Ao buscar uma lista de restaurantes, como em 6.10. Entendendo o Eager Loading, as entidades contidas em restaurantes também serão buscadas pelo banco de dados, ex: 1 Restaurante tem 1 Cozinha, que contém 1 Cidade, que contém 1 Estado e assim sempre que conter como propriedades outras entidades, para alterar esse comportamento, podemos utilizar uma propriedade da anotação @ManyToOne. Lembrando que em relacionamentos ToOne, a estratégia padrão é EAGER. Diferente da estratégia LAZY que só faz a busca no banco de dados se necessário dentro do código (serialização ou utilização dos métodos da entidade associada).


```
@ManyToOne(fetch = FetchType.LAZY)
@JsonIgnore
@JoinColumn(name = "endereco_cidade_id")
private Cidade cidade;
```

Como Muitos Restaurantes possuem Uma Cozinha , o relacionamento é de ManyToOne e relacionamentos ToOne utilizam a estratégia EAGER para buscar as entidades associadas mesmo que não utilizadas. Podemos utilizar a propriedade fetch atribuindo o tipo de "estratégia de busca". Para mudar estratégias ToOne é utilizado o oposto do padrão EAGER, utilizamos LAZY (fetch=FetchType.LAZY)

- Detalhe: mesmo ignorando propriedades para serialização com @JsonIgnore, a estratégia EAGER busca as entidades associadas.

Requisição listando todas os restaurantes:

```
2023-02-20 13:57:03.550  INFO 12172 --- [nio-8080-exec-2] o
Hibernate:
    select
        restaurant0_.id as id1_8_,
        restaurant0_.cozinha_id as cozinha11_8_,
        restaurant0_.data_atualizacao as data_atu2_8_,
        restaurant0_.data_cadastro as data_cad3_8_,
        restaurant0_.endereco_bairro as endereco4_8_,
        restaurant0_.endereco_cep as endereco5_8_,
        restaurant0_.endereco_cidade_id as enderec12_8_,
        restaurant0_.endereco_complemento as endereco6_8_,
        restaurant0_.endereco_logradouro as endereco7_8_,
        restaurant0_.endereco_numero as endereco8_8_,
        restaurant0_.nome as nome9_8_,
        restaurant0_.taxa_frete as taxa_fr10_8_
    from
        restaurante restaurant0_
```

Agora em vez de 6 selects como em 6.10. Entendendo o Eager Loading, temos apenas 1, pois a estratégia está ignorando entidades associadas como Cozinha e Cidade.

Ao tentar fazer uma requisição serializando essas entidades LAZY, temos um erro: (não ignorando as propriedades com JsonIgnore)


```
"timestamp": "2023-02-20T17:06:57.603+00:00",
"status": 500,
"error": "Internal Server Error",
"trace": "org.springframework.http.converter.HttpMessageConversionException: Type definition error: [simple type,
    class org.hibernate.proxy.pojo.bytebuddy.ByteBuddyInterceptor]; nested exception is com.fasterxml.jackson.
    databind.exc.InvalidDefinitionException: No serializer found for class org.hibernate.proxy.pojo.bytebuddy.
    ByteBuddyInterceptor and no properties discovered to create BeanSerializer (to avoid exception, disable
    SerializationFeature.FAIL_ON_EMPTY_BEANS) (through reference chain: java.util.ArrayList[0]->com.algaworks.
    algafood.domain.model.Restaurante[\"cozinha\"]->com.algaworks.algafood.domain.model.
    Cozinha$HibernateProxy$C4x6e0aO[\"hibernateLazyInitializer\"])\r\n\tat org.springframework.http.converter.
    json.AbstractJackson2HttpMessageConverter.writeInternal(AbstractJackson2HttpMessageConverter.java:462)
```


indicando que o Jackson não conseguiu serializar uma propriedade na classe/proxy temporária (do Hibernate) chamada hibernateLazyInitializer, pois na estratégia LAZY, as entidades só serão chamadas caso necessário, e na hora da serialização, as entidades não estão instanciadas, para não ter erro de NullPointerExeption, o próprio Hibernate cria um proxy encapsulando a entidade LAZY.

Para mudar esse comportamento, anotamos com @JsonIgnoreProperties e passando no valor o nome da propriedades, também é aceito múltiplos valores com {}, na propriedade de uma entidade LAZY para ignorar uma propriedade de dentro da entidade, pois não temos acesso a classe proxy criada em tempo de execução.

```java
@ManyToOne(fetch = FetchType.LAZY)
//@JsonIgnore
@JsonIgnoreProperties("hibernateLazyInitializer")
@JoinColumn(name = "cozinha_id", nullable = false)
private Cozinha cozinha;
```


```
Hibernate: select restaurant0_.id as id1_8_, restaurant0_.cozinha_id as cozinha11_8_, restaurant0_.data_atualizacao as dat
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome2_1_0_ from cozinha cozinha0_ where cozinha0_.id=?
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome2_1_0_ from cozinha cozinha0_ where cozinha0_.id=?
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome2_1_0_ from cozinha cozinha0_ where cozinha0_.id=?
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome2_1_0_ from cozinha cozinha0_ where cozinha0_.id=?
```

No log da aplicação, existem 6 consultas ao banco com LAZY nas entidades associadas como em 6.10. Entendendo o Eager Loading, aparentemente o comportamento é o mesmo da estratégia EAGER, porém, a consulta foi feita por conta da serialização das entidades na representação do recurso, e por isso foi feito a consulta nas entidades cozinha menos em cidade, pois está anotada com LAZY no fetchType.

Caso precisasse de dados da cidade de um restaurante, poderia chamar algum método da entidade cidade para o Hibernate buscar no banco, como em:

```
Hibernate: select restaurant0_.id as id1_8_, restaurant0_.cozinha_id as cozinha11_8_, restaurant0_.data_atualizacao as d
Hibernate: select cidade0_.id as id1_0_0_, cidade0_.estado_id as estado_i3_0_0_, cidade0_.nome as nome2_0_0_, estado1_.i
Uberlândia
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome2_1_0_ from cozinha cozinha0_ where cozinha0_.id=?
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome2_1_0_ from cozinha cozinha0_ where cozinha0_.id=?
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome2_1_0_ from cozinha cozinha0_ where cozinha0_.id=?
Hibernate: select cozinha0_.id as id1_1_0_, cozinha0_.nome as nome2_1_0_ from cozinha cozinha0_ where cozinha0_.id=?
```

```java
@GetMapping
public ResponseEntity<List<Restaurante>> listar() {
    final List<Restaurante> restaurantes = restauranteService.listar();


    System.out.println(restaurantes.get(0).getEndereco().getCidade().getNome());

    return ResponseEntity.ok(restaurantes);
}
```

6.13. Alterando a estratégia de fetching para Eager Loading

segunda-feira, 20 de fevereiro de 2023 17:20

Para relacionamentos ToMany, a estratégia de fetch padrão é LAZY, podemos alterar para EAGER

```
@ManyToMany(fetch = FetchType.EAGER)
@JoinTable(name = "restaurante_forma_pagamento",
           joinColumns = @JoinColumn(name = "restaurante_id"),
           inverseJoinColumns = @JoinColumn(name = "forma_pagamento_id"))
private List<FormaPagamento> formasPagamento = new ArrayList<>();
```

O Hibernate fez o select em todas as entidades que utilizam EAGER como fetchType

```
Hibernate: select restaurant0_.id as id1_8_, restaurant0_.cozinha
Hibernate: select formaspaga0_.restaurante_id as restaura1_9_0_,
Hibernate: select formaspaga0_.restaurante_id as restaura1_9_0_,
Hibernate: select formaspaga0_.restaurante_id as restaura1_9_0_,
Hibernate: select formaspaga0_.restaurante_id as restaura1_9_0_,
Hibernate: select formaspaga0_.restaurante_id as restaura1_9_0_,
Hibernate: select formaspaga0_.restaurante_id as restaura1_9_0_,
```

Para cada restaurante o Hibernate faz o select das formas de pagamento

É raro alterar um relacionamento ToMany LAZY ser alterado para EAGER

6.14. Resolvendo o Problema do N+1 com fetch join na JPQL

segunda-feira, 20 de fevereiro de 2023 18:32

Para resolver o problema do fetch padrão de relacionamentos ToOne EAGER, que faz consultas em todas as entidades associadas gerando várias consultas para cada entidade associada, como em 6.10. Entendendo o Eager Loading, podemos mudar o comportamento do Hibernate.

É um problema pois, para cada Restaurante, temos uma Cozinha e Cidade associados, mesmo que não usássemos os dados das entidades associadas, o Hibernate faz consultas pesadas e muitas vezes desnecessárias ao banco.

A implementação padrão do repositório anotado com @Repository é implementado pelo Spring Data JPA, podemos criar nossas consultas JPQL com @Query em cima da assinatura do método na interface

```
@Query("from Restaurante as r join r.cozinha")
List<Restaurante> findAll();
```

saída:

```
2023-02-20 19:01:42.505  INFO 7024 --- [nio-8080-exec-2] o.s.web.servl
Hibernate:
    select
        restaurant0_.id as id1_8_0_,
        cozinha1_.id as id1_1_1_,
        restaurant0_.cozinha_id as cozinha11_8_0_,
        restaurant0_.data_atualizacao as data_atu2_8_0_,
        restaurant0_.data_cadastro as data_cad3_8_0_,
        restaurant0_.endereco_bairro as endereco4_8_0_,
        restaurant0_.endereco_cep as endereco5_8_0_,
        restaurant0_.endereco_cidade_id as enderec12_8_0_,
        restaurant0_.endereco_complemento as endereco6_8_0_,
        restaurant0_.endereco_logradouro as endereco7_8_0_,
        restaurant0_.endereco_numero as endereco8_8_0_,
        restaurant0_.nome as nome9_8_0_,
        restaurant0_.taxa_frete as taxa_fr10_8_0_,
        cozinha1_.nome as nome2_1_1_
    from
        restaurante restaurant0_
    inner join
        cozinha cozinha1_
            on restaurant0_.cozinha_id=cozinha1_.id
```

O Hibernate faz somente 1 select, uma ida ao banco de dados trazendo apenas o necessário.

Podemos fazer em mais entidades, especialemente para associações toMany

```java
@Query("from Restaurante as r join fetch r.cozinha")
List<Restaurante> findAll();
```

```
Hibernate:
    select
        restaurant0_.id as id1_8_0_,
        cozinha1_.id as id1_1_1_,
        restaurant0_.cozinha_id as cozinha11_8_0_,
        restaurant0_.data_atualizacao as data_atu2_8_0_,
        restaurant0_.data_cadastro as data_cad3_8_0_,
        restaurant0_.endereco_bairro as endereco4_8_0_,
        restaurant0_.endereco_cep as endereco5_8_0_,
        restaurant0_.endereco_cidade_id as enderec12_8_0_,
        restaurant0_.endereco_complemento as endereco6_8_0_,
        restaurant0_.endereco_logradouro as endereco7_8_0_,
        restaurant0_.endereco_numero as endereco8_8_0_,
        restaurant0_.nome as nome9_8_0_,
        restaurant0_.taxa_frete as taxa_fr10_8_0_,
        cozinha1_.nome as nome2_1_1_
    from
        restaurante restaurant0_
    inner join
        cozinha cozinha1_
            on restaurant0_.cozinha_id=cozinha1_.id
```

Mas acabamos criando um produto cartesiano onde se repete registro, mas o Hibernate não deixa duplicar os resultados, é bom deixar fetch até onde não precisa.