O Hibernate é uma das especificações mais utilizadas da JPA, e fornece diversas anotações para a utilização do Mapeamento Objeto-Relacional.

Vamos conhecer as principais delas?

**@Entity**

Essa anotação é usada para marcar uma classe como uma entidade que deve ser mapeada para uma tabela de banco de dados. Cada entidade corresponde a uma tabela no banco de dados.

**@Table**

Por padrão, o Hibernate usa o nome da classe como o nome da tabela no banco de dados, fazendo apenas a conversão de padrão de nomenclatura do PascalCase para o SnakeCase, que é o padrão utilizado no banco de dados, no entanto, caso seja necessário que o nome da classe seja diferente do nome da tabela no banco de dados, é possível utilizar esta anotação que permite personalizar o mapeamento entre a classe de entidade e a tabela de banco de dados. Com ela, você pode especificar o nome da tabela, o esquema e as restrições de chave primária.

@Entity

@Table(name = "minha\_tabela")

public class MinhaEntidade { ... }

Copiar código

**@Id**

Marca um campo como a chave primária da entidade. O Hibernate usa essa anotação para identificar exclusivamente os registros no banco de dados.

**@GeneratedValue**

Usada em conjunto com @Id, essa anotação especifica como a chave primária é gerada automaticamente. Pode ser usada com estratégias como AUTO, IDENTITY, SEQUENCE ou TABLE, dependendo do banco de dados.

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

Copiar código

**@Column**

Similar ao que acontece na anotação @Table, o Hibernate utiliza o nome dos atributos e os converte como sendo idênticos aos nomes das colunas no banco de dados, e caso seja necessário utilizar nomes diferentes,você pode configurar o nome da coluna, bem como seu tipo, e se ela é obrigatória.

@Column(name = "nome\_completo", nullable = **false**)

**private** **String** nome;

Copiar código

**@OneToMany e @ManyToOne**

Usadas para mapear relacionamentos de um-para-muitos e muitos-para-um entre entidades. Elas definem as associações entre as tabelas no banco de dados.

@Entity

public class Autor {

@OneToMany(mappedBy = "autor")

private List<Livro> livros;

}

@Entity

public class Livro {

@ManyToOne

@JoinColumn(name = "autor\_id")

private Autor autor;

}

Copiar código

**@ManyToMany**

A anotação @ManyToMany é usada para mapear relacionamentos muitos-para-muitos entre entidades.

**@OneToOne**

A anotação @OneToOne é usada para mapear relacionamentos um-para-um entre entidades.

**@JoinColum**

A anotação @JoinColumn é usada para especificar a coluna que será usada para representar um relacionamento entre entidades. É frequentemente usada em conjunto com @ManyToOne e @OneToOne.

@ManyToOne

@JoinColumn(name = "autor\_id")

private Autor autor;

Copiar código

**@JoinTable**

A anotação @JoinTable é usada para mapear tabelas de junção em relacionamentos muitos-para-muitos. Ela especifica a tabela intermediária que liga duas entidades.

@Entity

public class Estudante {

@ManyToMany

@JoinTable(name = "inscricao",

joinColumns = @JoinColumn(name = "estudante\_id"),

inverseJoinColumns = @JoinColumn(name = "curso\_id"))

private List<Curso> cursos;

}

Copiar código

**@Transient**

A anotação @Transient é usada para marcar uma propriedade como não persistente. Isso significa que a propriedade não será mapeada para uma coluna no banco de dados.

@Transient

**private** transientProperty;

Copiar código

**@Enumerated**

A anotação @Enumerated é usada para mapear campos enumerados (enum) para colunas do banco de dados.

@Enumerated(EnumType.STRING)

**private** Status status;

Copiar código

**@NamedQuery**

Essa anotação é usada para definir consultas JPQL nomeadas que podem ser reutilizadas em várias partes do código.

@Entity

@NamedQuery(name = "Cliente.findAll", query = "SELECT c FROM Cliente c")

public class Cliente { ... }

Copiar código

**@Cascade**

A anotação @Cascade é usada para especificar o comportamento de cascata das operações de persistência, como salvar e excluir, em relacionamentos. Por exemplo, você pode configurar para que as operações de salvar em cascata afetem entidades relacionadas.

@OneToMany(mappedBy = "departamento")

@Cascade(CascadeType.SAVE\_UPDATE)

private List<Funcionario> funcionarios;

Copiar código

**@Embeddable e @Embedded**

Essas anotações são usadas para representar tipos incorporados (embeddable types) que podem ser usados como componentes em entidades.

@Embeddable

public class Endereco { ... }

@Entity

public class Cliente {

@Embedded

private Endereco endereco;

}

Copiar código

Além dessas anotações, há muitas outras que podem ser consultadas na [documentação de anotações do Hibernate](https://docs.jboss.org/hibernate/stable/annotations/reference/en/html/), e que facilitam muito o dia a dia de pessoas desenvolvedoras que usam o ORM.

**Para saber mais: estratégias de geração do ID**

Ao trabalharmos com tabelas em bancos de dados, as chaves primárias são essenciais. Geralmente, criamos um identificador único ou ID para representar essas chaves da melhor forma possível.

As chaves primárias servirão para que cada entidade seja salva de forma única. Se quando trabalhamos com Java a JVM se encarrega de separar cada objeto em um lugar da memória, no banco de dados isso precisa ser configurado. E utilizar um ID costuma ser uma boa forma de fazer isso.

Temos várias formas de gerar IDs no banco de dados, e a JPA também nos ajuda com isso. Basta utilizar a anotação GeneratedValue e escolher a estratégia de geração do ID. Posteriormente vamos falar sobre essas diversas estratégias existentes, ok?!

Confira:

**Estratégia AUTO**

A estratégia AUTO é a estratégia padrão de geração de ID JPA. Quando esta estratégia é utilizada, é a JPA que determina a melhor estratégia a ser usada, baseando-se no banco de dados específico que você está utilizando.

Aqui está um exemplo de como implementar a estratégia AUTO em um código:

@Entity

**public** **class** **Entidade** {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

**private** Long id;

}

Copiar código

Esta estratégia é bem prática, pois a JPA se encarrega de toda a lógica de geração de ID. No entanto, pode ser pouco flexível se você precisar de um maior controle sobre a geração de ID.

**Estratégia SEQUENCE**

A estratégia SEQUENCE utiliza um banco de dados de sequência para gerar IDs. Com essa estratégia, a sequência é incrementada cada vez que um ID é gerado, garantindo que cada ID seja único.

Aqui está um exemplo de como implementar a estratégia SEQUENCE:

@Entity

**public** **class** **Entidade** {

@Id

@GeneratedValue(strategy=GenerationType.SEQUENCE)

**private** Long id;

}

Copiar código

Essa estratégia é particularmente útil para bancos de dados que suportam sequências, pois é muito eficiente e evita a possibilidade de colisões de ID.

**Estratégia IDENTITY**

A estratégia IDENTITY faz uso de colunas de auto incremento disponibilizadas por alguns bancos de dados. Cada vez que uma nova entidade é inserida no banco de dados, a coluna de ID auto incrementada é atualizada para gerar um novo ID.

Exemplo de implementação desta estratégia:

@Entity

**public** **class** **Entidade** {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

**private** Long id;

}

Copiar código

Esta estratégia é útil se você estiver usando um banco de dados que suporta colunas de auto incremento, como MySQL ou SQL Server.

**Estratégia TABLE**

Por último, temos a estratégia TABLE. Essa estratégia utiliza uma tabela de banco de dados separada para gerar IDs. Ela é menos eficiente do que as outras estratégias, mas tem a vantagem de ser portátil entre diferentes provedores de banco de dados.

Exemplo de implementação desta estratégia:

@Entity

**public** **class** **Entidade** {

@Id

@GeneratedValue(strategy=GenerationType.TABLE)

**private** Long id;

}

Copiar código

Compreender essas estratégias e saber quando usar cada uma delas permitirá que você use a JPA de maneira mais eficiente, garantindo que seus dados sejam persistidos de maneira confiável e segura. Lembre-se, a escolha da estratégia de ID correta pode ter um grande impacto na eficiência da sua aplicação!