JPA(Java Persistence API)

Introdução:
JPA ajuda os programas feitos em Java a trabalharem facilmente com bancos de dados, que são como armazéns para guardar informações importantes. Em vez de lidar diretamente com as regras complicadas dos bancos de dados, a JPA permite que os programadores usem objetos para guardar e pegar informações desses bancos de dados.

Principais Conceitos:
- Mapeamento Objeto Relacional(ORM – Object-Relational Mapping)
  Permite mapear as entidades e seus relacionamentos (tabelas) em objetos que podem ser persistidos (entidades).
- Uma API (EntityManager)
  Permite desempenhar operações CRUD sobre um banco de dados(Criar, Ler, Atualizar e Remover).
- Uma linguagem de consulta (JPQL – Java Persistence Query Language)
  Permite recuperar dados com uma linguagem de consulta Orientada à Objetos.

Quando fazemos a correspondência entre objetos e um banco de dados relacional, usamos o termo "entidade":

- Objetos são apenas mantidos na memória do computador.
- Entidades são objetos que ficam na memória por um tempo e depois são guardados no banco de dados.
- Uma vez que as entidades estão no banco de dados, podemos fazer perguntas sobre elas usando a JPQL.

Mapeamento Objeto Relacional - ORM:
Ele disponibiliza ferramentas fundamentadas em conceitos de objetos, permitindo a execução e manipulação de comandos relacionados a dados por meio da linguagem de programação escolhida (que deve suportar a abordagem orientada a objetos).

ORM

Prós:
- É mais fácil usar o ORM para consultar do que o MySQL.
- Não há necessidade de alterar o código se o banco de dados subjacente for alterado.

Contras:
- Aprender um ORM do zero é demorado

- Você pode encontrar problemas de desempenho ao codificar consultas complexas usando ORM

Mapeamento Objeto Relacional - ORM:

-Table envolve o mapeamento entre objetos e tabelas em um banco de dados. A anotação `@Table` é usada para definir propriedades da tabela, como seu nome. Por padrão, os nomes das tabelas são criados com letras maiúsculas, semelhantes aos nomes das classes correspondentes.

-A anotação @SecondaryTable permite distribuir os atributos de uma classe em várias tabelas.
-Para várias tabelas secundárias, utilize @SecondaryTables.
-Colunas não atribuídas a uma tabela específica serão mapeadas na tabela principal.
-É importante considerar questões de desempenho ao optar por usar tabelas secundárias.

Chaves Primárias:
-A anotação @Id define uma chave primária.
-Tipos aceitáveis incluem primitivos (byte, int, short, long e char), classes "wrapper" (Byte, Integer, Short, Long, Character), Strings, números e datas (String, BigInteger, Date).
-@GeneratedValue indica que a coluna tem um valor gerado automaticamente.
-O atributo "strategy" define a estratégia de geração de valores incrementados:
- SEQUENCE e IDENTITY: para colunas sequence ou identity, respectivamente.
- TABLE: usa uma tabela para armazenar a "semente" da sequência.
- AUTO: escolha automática da estratégia de geração de chave.

Atributos:
- @Basic é um mapeamento simples que define a obrigatoriedade (opcional) e a forma de carregamento (fetch) do atributo.
- @Column define várias propriedades comuns das colunas em banco de dados, como nome, comprimento, unicidade etc.

Outras anotações incluem:
- @Temporal para definir datas (Date, Time e Timestamp).
- @Transient permite que um atributo não seja persistido.
- Lembre-se: as anotações de mapeamento de atributos também podem ser usadas nos métodos 'get'.

Associações no JPA seguem os princípios das associações de objetos em Java. Elas são naturalmente unidirecionais e não são gerenciadas pelo contêiner.

Essas associações representam relacionamentos entre entidades, seja diretamente ou através de coleções. Existem três tipos principais de associações no JPA:

- @OneToOne: Indica uma relação um-para-um entre entidades.
- @ManyToOne: Define uma relação muitos-para-um, onde várias entidades se relacionam com uma única entidade.
- @ManyToMany: Representa uma relação muitos-para-muitos, onde várias entidades podem se relacionar com várias outras entidades.