**Spring Data JPA**

É mais um projeto do ecossistema Spring utilizado para trabalhar com base de dados em nossas aplicações Spring de forma mais simples.

Mas antes disso, vamos entender alguns termos/camadas quando se trata da relação entre a base de dados e nossa aplicação:

- **Spring Data JPA**: é uma camada de abstração adicional que inclui todos os recursos da especificação JPA, além dos recursos próprio, como a implementação do padrão de repositories, criação de consultas na base de dados a partir de nomes de atributos. Ele engloba todo o JPA.

- **JPA (Java Persistence API)**: é uma especificação para o mapeamento objeto-relacional em Java, ou seja, é a forma de lidarmos com todo esse mapeamento das entidades que vamos criando em forma de código para transformar tudo isso em tabelas, colunas na base de dados, podendo fazer tudo isso incluindo Notations, consultas JPQL, API’s para realizar toda essa interação com a base de dados. Ele é uma abstração, logo ele precisa de alguém que implemente-o, tendo com o Hibernate pra fazer essa função.

- **Hibernate**: É uma das implementações mais comuns e utilizadas na especificação JPA. Usando assim o JDBC por de baixo dos panos para fazer todas as transições de uma base de dados, para que seja possível iniciar as conexões, executas todas as transações/query’s.

**JPA Notations:**

Auxilia todo o mapeamento do objeto-relacional para a base de dados.

**@Entity**: Indica que a classe é uma entidade que será mapeada para uma tabela no banco de dados.

**@Table**: Especifica detalhes adicionais sobre a tabela correspondente à entidade, como o nome da tabela no banco de dados.

**@Id**: Define o atributo como chave primária da tabela.

**@GeneratedValue**: Configura como o valor da chave primária é gerado automaticamente, por exemplo, usando uma estratégia de autoincremento.

**@Column**: Mapeia um atributo da classe para uma coluna específica na tabela do banco de dados. Permite configurar detalhes como nome da coluna, nullable, tamanho máximo, entre outros.

**@Transactional**: Indica que o método deve ser executado dentro de uma transação. Isso é usado para garantir atomicidade das operações no banco de dados.

**@JoinTable**: Utilizado em relacionamentos muitos-para-muitos para especificar a tabela intermediária que será usada para unir as duas entidades.

**@JoinColumn**: Define a coluna na tabela atual que é a chave estrangeira para a entidade relacionada.

Anotações relacionadas ao tipo de relacionamento entre entidades:

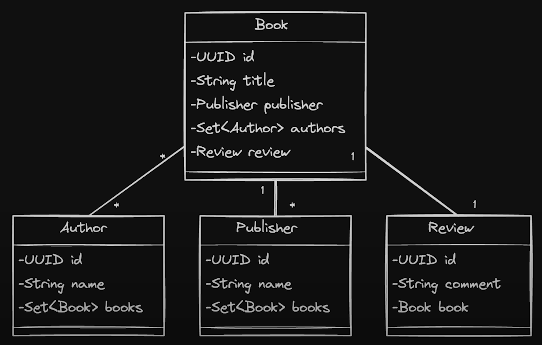
**@ManyToMany**: Define um relacionamento muitos-para-muitos entre duas entidades. Isso requer uma tabela intermediária.

**@ManyToOne**: Define um relacionamento muitos-para-um entre duas entidades. O lado "muitos" é a entidade atual.

**@OneToMany**: Define um relacionamento um-para-muitos entre duas entidades. O lado "muitos" é a entidade relacionada.

**@OneToOne**: Define um relacionamento um-para-um entre duas entidades. Cada entidade pode estar relacionada a no máximo uma instância da outra entidade.

**Projeto que vamos implementar o Spring Data JPA.**

****

**Conexão com o banco de dados:**

Toda a conexão com o BD vai ser feita no application.properties.

spring.application.name=jpa  
  
spring.datasource.url= jdbc:postgresql://localhost:5432/bookstore-jpa  
spring.datasource.username=postgres  
spring.datasource.password=victormacedo  
spring.jpa.hibernate.ddl-auto=update  
  
spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation=true  
spring.jpa.show-sql=true

- Definimos a URL de conexão (qual banco de dados a gente vai usar, endereço local que está instalado essa base de dados na minha máquina e o nome dela).

- Passamos algumas credenciais utilizadas na instalação do postgres (username, password)

- Sempre que iniciamos a aplicação Spring boot, automaticamente refletimos tudo que estamos representando em código Java para a base de dados. Ou seja, se criamos uma entidade em Java, criamos uma tabela na base de dados, a mesma coisa quando criamos atributos na entidade, criamos colunas na base de dados.

- Definimos que o lob do jdbc seja true, pois algumas vezes o hibernate ele vai buscar metadados do postgres e caso os metadados não esteja disponível naquele momento, ele vai gerar log de vários erros em nosso console.

- Vermos todo o SQL que está sendo gerado no nosso console todas as vezes que ele for criado, modificado e etc.

**Mapeando as entidades JPA**

Dentro do pacote principal (com.bookstore.jpa), nós criamos um outro pacote chamado “models” onde vamos inserir nossos models que vamos implementar.

Para isso, utilizamos algumas notations, sendo elas:

**@Entity** – Declaramos que essa classe é uma entidade JPA.

**@Table(name = “TB\_BOOK”)** – declaramos qual vai ser o nome dessa entidade JPA na base de dados.

**implements Serializable** – Interface de marcação para mostrarmos para a JVM que essa classe pode ser serializada e definimos também o serialVersionUID para essas serializações que forem feitas.