¿Qué es Object-Relational mapping (ORM)?

Es un modelo de programación que permite mapear las estructuras de una base de datos relacional. Este mecanismo describe cómo se almacena un objeto y sus relaciones en un conjunto de tablas en una base de datos. Existen estándares y frameworks para todos los lenguajes modernos, por ej, Laravel cuyo ORM es "Eloquent" y .NET Core cuyo ORM se llama Entity Framework Core.

Características a tener en cuenta:

- Los objetos persistentes se denominan “Entity” o entidades.
- El objetivo principal es no perder las ventajas de la orientación a objetos al interactuar con una base de datos.
- El mapeo se configura a través de metadatos (mediante archivos xml o anotaciones en las clases) Mapeo Objeto-Relacional.

Ventajas de utilizar un ORM:

- Reduce la cantidad de código necesario para lograr la persistencia y permite que la aplicación sea más fácil de mantener.
- Hace más sencillo el acceso a la base de datos.
- Habrá consultas que tendrán mejor rendimiento que si fueran hechas con SQL común.
- Proporciona más seguridad ya que los ORM tienen protección contra inyecciones SQL y otros tipos de ataques.

Desventajas:

- Requiere mantener metadatos de los objetos a persistir.

¿Qué es JPA?

La persistencia de datos es un medio mediante el cual una aplicación puede recuperar información desde un sistema de almacenamiento no volátil y hacer que esta persista.
Es una abstracción por sobre JDBC que permite realizar en forma sencilla una correlación entre objetos y registros de base de datos.
JPA es una especificación de Java EE basada en interfaces que son implementadas por diferentes proveedores de persistencia como Hibernate o Eclipselink entre otros.

Dado que es una especificación, JPA no proporciona clase alguna para poder trabajar con la información. Lo que hace es proveernos de una serie de interfaces que podemos utilizar para implementar la capa de persistencia de nuestra aplicación, apoyándonos en alguna implementación concreta de JPA.

JPA requiere meta-información en las clases Java para poder realizar el mapeo objeto-relacional.

Se pueden añadir metadatos de dos maneras:

- Archivos XML
- Anotaciones .

¿Qué diferencia existe entre JPA y Hibernate? Mencione otras implementaciones.

Java Persistence API, más conocida por sus siglas JPA, es la API de persistencia desarrollada para la plataforma Java EE . La cual es un framework del lenguaje de programación Java, que maneja datos relacionales en aplicaciones usando la plataforma Java en sus ediciones Standard (Java SE) y Enterprise (Java EE) .

Por su parte, Hibernate es una herramienta de Mapeo objeto-relacional (ORM), para la plataforma Java – también disponible para .Net, bajo el nombre de NHibernate – que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación; mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones . Este es software libre, distribuido bajo los términos de la licencia GNU LGPL.

Es importante destacar que JPA es una parte de la especificación de EJB 3, es decir que no es un framework, sino que es simplemente un documento en el cual se especifica los principios básicos de gestión de la capa de persistencia en el mundo de Java EE . En cambio, Hibernate, si se trata de un framework que gestiona la capa de persistencia a través de ficheros xml o anotaciones.

Existen más implementaciones de JPA que al fin y al cabo es únicamente una especificación, que son las siguientes

EclipseLink : Otra implementación muy utilizada ya que esta apoyada por la fundación Eclipse y da soporte a JPA.

DataNucleus : Otro framework de persistencia que soporta JPA pero incluye muchas más opciones y tipos de persistencia.

TopLink : La implementación de Oracle clásica para sus productos hoy por hoy esta basada en EclipseLink

ObjectDB: Otra implementación de JPA que promete un rendimiento alto comparado con sus competidores.

Apache OpenJPA.

Describa el propósito del EntityManager y de la unidad de persistencia (Persistence Unit)

Un **Entity Manager** gestiona las entidades monitoreando los cambios realizados al estado de los objetos. El conjunto de entidades gestionadas por un Entity Manager se conoce como contexto de persistencia. Todas las operaciones relacionadas con la persistencia de las entidades se realizan a través de un gestor de entidades (entity manager en inglés). El funcionamiento del entity manager está especificado en una única interfaz llamada EntityManager.

El **entity manager** tiene dos responsabilidad fundamentales:

Define una conexión transaccional con la base de datos que debemos abrir y mantener abierta mientras estamos realizado operaciones. En este sentido realiza funciones similares a las de una conexión JDBC.

Además, mantiene en memoria una caché con las entidades que gestiona y es responsable de sincronizarlas correctamente con la base de datos cuando se realiza un flush. El conjunto de entidades que gestiona un entity manager se denomina su contexto de persistencia.

Unidad de persistencia

Define un modelo relacional de objetos completo que correlaciona clases Java (entidades + estructuras de soporte) con una base de datos relacional. EntityManagerFactory utiliza estos datos para crear un contexto de persistencia al que se puede acceder mediante EntityManager.

Describa las siguientes anotaciones de JPA: @Entity, @Table, @Column, @Id, @GeneratedValue

@Entity: se utiliza para definir una clase como una entidad en la base de datos.
@Table: se utiliza para definir una clase como tabla en la base de datos.
@Column: se utiliza para definir un atributo como una columna sobre una tabla.
@Id:Se utiliza para definir un atributo como clave primaria en una tabla.
@GeneratedValue:Se utiliza para definir cómo se va a generar la clave primaria de la tabla.

Explique los distintos tipos de relaciones y los distintos tipos de mapeo de JPA. Describa las anotaciones @JoinColumn y @JoinTable

En JPA existen distintos tipos de relaciones entre las entidades:

Uno a Uno (One to One). Utiliza el decorador @OneToOne. En esta relación, cada entidad está asociada directamente con una única entidad.

Uno a Muchos (One to Many). Utiliza el decorador @OneToMany. Cada entidad puede estar asociada con muchas.Normalmente esta asociación se fija con un objeto de tipo List, Set, Map, SortedSet o SortedMap.

Muchos a Uno (Many to One). Se identifica con el decorador @ManyToOne. Es justo la relación opuesta a Uno a Muchos. Normalmente se concreta mediante el uso de un mapeo bidireccional.

Muchos a Muchos (Many to Many). Se identifica mediante el decorador @ManyToMany. En él, varias entidades está relacionadas a su vez con varias entidades. Cada entidad tiene un objeto de tipo List o Set, que hace referencia a la otra entidad. Esto resulta en una tabla de unión en la que se definen las relaciones

@JoinColumn: Esta anotación especifica la columna de clave externa para la tabla de vinculación, o el nombre de columna de la tabla de destino de vinculación a la que se hace referencia desde la columna de clave externa, cuando las clases de entidad están correlacionadas.

@JoinTable: Esta anotación especifica la configuración de la tabla de vinculación en las siguientes clases:

- Clase de propietario cuando se especifica la relación ManyToMany.
- Clase que contiene una relación OneToMany de una sola dirección.

¿Cuáles son las distintas estrategias que existen en JPA para mapear relaciones de herencia?

Las estrategias que podemos utilizar para la herencia son SINGLE_TABLE, JOINED y TABLE_PER_CLASS:

SINGLE_TABLE: Mapea toda la jerarquia a una sola tabla, con todos los atributos existentes en la superclase y en las subclases. Definimos la estrategia, con la annotation @Inheritance en la superclase.

JOINED: Ésta estrategia genera una tabla por cada clase de la jerarquía, sea abstracta o concreta. Es decir que cada atributo de las clases van a estar en su tabla correspondiente de la base de datos, y mediante JOINS entre las subclases y la superclases se pueden obtener los objetos.

TABLE_PER_CLASS: Se genera una tabla por cada subclase de la jerarquía, repitiendo los atributos de la superclase, en cada tabla que representan a las subclases.

¿Qué es Spring Data y, más particularmente, Spring Data JPA?

Spring Data JPA, parte de la familia Spring Data más grande, facilita la implementación de repositorios basados en JPA. Este módulo trata sobre el soporte mejorado para las capas de acceso a datos basadas en JPA. Facilita la creación de aplicaciones impulsadas por Spring que utilizan tecnologías de acceso a datos.

La implementación de una capa de acceso a datos de una aplicación ha sido engorrosa durante bastante tiempo. Se debe escribir demasiado código repetitivo para ejecutar consultas simples, así como para realizar la paginación y la auditoría. Spring Data JPA tiene como objetivo mejorar significativamente la implementación de capas de acceso a datos al reducir el esfuerzo a la cantidad que realmente se necesita. Como desarrollador, escribe las interfaces de su repositorio, incluidos los métodos de búsqueda personalizados, y Spring proporcionará la implementación automáticamente.

¿Cuál es el objetivo de la capa de abstracción de Repositorios? Describa la anotación @Repository y las interfaces JPARepository y CRUDRepository.

Las capas de abstracción son componentes de software que permiten encapsular funcionalidades de un sistema permitiendo la reutilización de componentes a través de una interfaz de programación estandarizada.

La interfaz central en la abstracción del repositorio de Spring Data es Repository(probablemente no una gran sorpresa). Se necesita la clase de dominio para administrar, así como el tipo de identificación de la clase de dominio como argumentos de tipo. Esta interfaz actúa principalmente como una interfaz de marcador para capturar los tipos con los que trabajar y para ayudarlo a descubrir interfaces que amplían esta. Proporciona CrudRepository una funcionalidad CRUD sofisticada para la clase de entidad que se administra.

Describa Query Methods. Menciones algunas subject keywords y predicate keywords. Describa la anotación @Query.

El proxy del repositorio tiene dos formas de derivar una consulta específica de la tienda a partir del nombre del método. Puede derivar la consulta del nombre del método directamente o mediante una consulta creada adicionalmente. Las opciones disponibles

dependen de la tienda real. Sin embargo, tiene que haber una estrategia que decida qué consulta real se crea.

@Query: Para definir SQL para ejecutar para un método de repositorio Spring Data, podemos anotar el método con la anotación @Query : su atributo de valor contiene el JPQL o SQL para ejecutar.