educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 3 - Java Architect

Módulo 30

ORM

¿Qué es la ORM?

Definición

ORM significa *Object / Relational Mapping*. Es el *middleware* que maneja la persistencia en la capa de acceso a datos.

Tiene como objetivo **manejar la persistencia de objetos de una aplicación**, dentro de una base de datos, en forma automatizada y transparente. Utiliza metadatos para describir la **relación entre el modelo de clases y el modelo de tablas.**

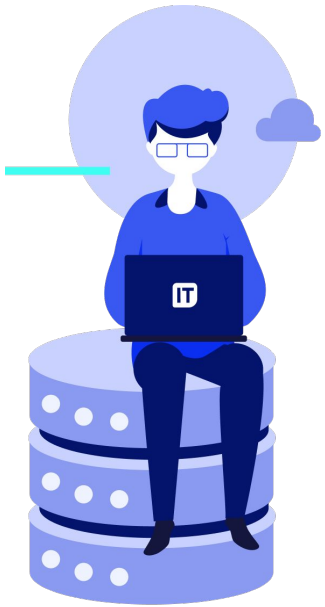
Maneja los casos más comunes, que son los que aparecen en mayor cantidad. Los casos especiales deben ser manejados por el usuario, aunque éstos deberían ser los mínimos.

El uso de un ORM suele disminuir el tiempo de construcción de una aplicación en un 30%. Se estima que el trabajo de desarrollo de la capa de acceso a datos disminuye en un 85%.

Organización

Una **solución del tipo ORM** tiene como partes a:

1. Una API para realizar los ABMC en objetos de clases persistentes.
2. Un lenguaje que permite realizar consultas referenciando a clases y atributos.
3. Una forma de especificar los mapeos entre metadatos.



Ventajas

Las ventajas que provee la utilización de un ORM son las siguientes:

- **Independencia de la base de datos**: provee una abstracción de la base de datos utilizada, lo que permite - en caso de no utilizar SQL propietario – una fácil migración.
- **Productividad**: reduce significativamente el tiempo de desarrollo.
- **Fácil mantenimiento**: agrega simplicidad al código, lo que hace que el mantenimiento sea simplificado.
- **Menor trabajo**: disminuye de forma importante la cantidad de líneas de código de la aplicación, ya que la mayoría del código ya está pre-escrito.

Tecnologías ORM

User-defined DAOs

DAO significa *Data Access Object*. Es un **patrón de diseño** utilizado para **realizar operaciones contra una base de datos,** pero abstrayendo de su lógica. La lógica de acceso a datos queda encapsulada dentro de una DAO, con lo cual no es visible para usuarios que utilizan el DAO.

El ***User-defined DAO*** es un DAO definido por el usuario. El desarrollador puede armar su propio modelo de acceso a datos con la construcción de diversos DAOs, formando así la ***DAL*** (*Data Access Layer*, o capa de acceso a datos).

Es una de las opciones más utilizadas para la persistencia de datos. En Java puntualmente, se trabaja con SQL y JDBC directamente. No se necesita ningún contenedor especial para utilizarlos. Son clases planas con funcionalidad. Tiene como desventaja que se paga el precio de "reinventar la rueda", debido a que hay que realizar el 100% de la codificación.

JPA (Java Persistence API)

Es una **API de persistencia** desarrollada para la plataforma **JAVA EE**.

Todos los componentes de la API se encuentran definidos en el paquete **`javax.persistence`**.

JPA es una especificación, lo que significa que por sí sola no se puede implementar. Podemos utilizar *annotations* de JPA en nuestras clases, sin embargo sin una implementación nada sucederá.

Hibernate es una implementación de JPA, cumple las especificaciones de JPA y además aporta sus propias *annotations* y funcionalidades.

Cuando se utilizan las *annotations* de JPA con Hibernate en realidad se está utilizando la implementación de Hibernate para las *annotations* de JPA. El beneficio es que podemos cambiar el ORM que estamos utilizando por cualquier otro que implemente JPA sin tener que modificar nuestro código.

Algunas **implementaciones de JPA** son:

- Hibernate.
- ObjectDB.
- TopLink.
- CocoBase.
- EclipseLink.
- OpenJPA.
- Kodo.
- DataNucleus, antes conocido como JPOX.
- Amber.

¡Sigamos trabajando!