MAPEO OBJETO RELACIONAL

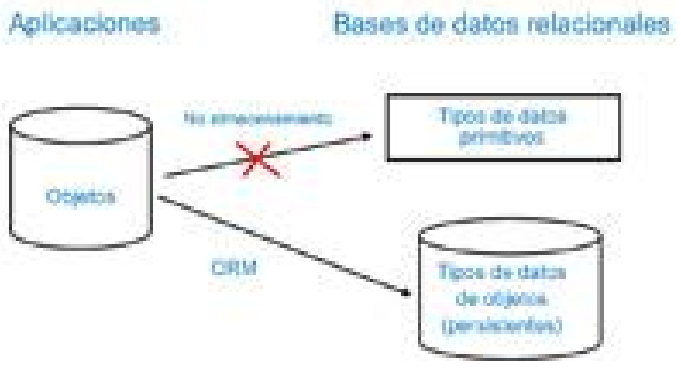
Introducción

En un sistema de bases de datos relacionales, la información se almacena en tablas con filas y columnas, mientras que en la programación orientada a objetos, los datos se organizan en objetos con atributos y métodos. El objetivo del ORM es proporcionar una capa de abstracción que permita a los desarrolladores trabajar con objetos en su código en lugar de lidiar directamente con las tablas y consultas de la base de datos.

¿Qué es?

Es una técnica de programación que se utiliza con el propósito de convertir datos entre el utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, **gracias a la persistencia.** Esto posibilita el uso en las bases de datos relacionales de las características propias de la programación orientada a objetos (básicamente herencia y polimorfismo).
Las bases de datos más extendidas son del tipo relacional y éstas sólo permiten guardar tipos de datos primitivos (enteros, cadenas de texto, etc.) por lo que no se puede guardar de forma directa los objetos de la aplicación en las tablas. Por tanto, se debe convertir los valores de los objetos en valores simples que puedan ser almacenados en una base de datos (y poder recuperarlos más tarde).

El mapeo objeto-relacional surge, pues, para dar respuesta a esta problemática: traducir los objetos a formas que puedan ser almacenadas en bases de datos preservando las propiedades de los objetos y sus relaciones; estos objetos se dice entonces que son persistentes.



Object Relational Mapping (ORM) es la herramienta que nos sirve para transformar representaciones de datos de los Sistemas de Bases de Datos Relacionales, a representaciones (Modelos) de objetos. Dado a que los RDBMS carecen de la flexibilidad para representar datos no escalables, l**a existencia de un ORM es fundamental para el desarrollo de sistemas de software robustos y escalables.**

En el modelo relacional, cada fila en la tabla se mapea a un objeto y cada columna a una propiedad.

Características

- Las herramientas ORM facilitan el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.
- Podemos conectar con una base de datos relacional para extraer la información contenida en objetos de programas que están almacenados. Para ello, sólo tendremos que definir la forma en la que establecer **la correspondencia entre las clases y las tablas** una sola vez (indicando qué propiedad se corresponde con cada columna, qué clase con cada tabla, etc.). Una vez hecho esto, podremos utilizar instancias de nuestra aplicación e indicar a la ORM que los haga persistentes, consiguiendo que una sola herramienta pueda leer o escribir en la base de datos.

Ventajas de ORM.

Ayudan a reducir el tiempo de desarrollo de software. La mayoría de las herramientas ORM disponibles, permiten la creación del modelo a través del esquema de la base de datos, es decir, el usuario crea la base de datos y la herramienta automáticamente lee el esquema de tablas y relaciones y crea un modelo ajustado.
Las ventajas del uso de un ORM incluyen:

- **Simplificación del código**: Los desarrolladores pueden trabajar con objetos y clases en lugar de escribir consultas SQL directamente, lo que simplifica el código y lo hace más legible.

- **Portabilidad:** Un ORM puede ser configurado para trabajar con diferentes sistemas de bases de datos sin cambiar el código de la aplicación, lo que facilita la portabilidad entre bases de datos.
-
- **Mantenimiento:** El ORM facilita la tarea de realizar cambios en la estructura de la base de datos, ya que los cambios en las clases de objetos se reflejan en las tablas de la base de datos de manera automática.
- **Seguridad:** Los ORM suelen incluir medidas de seguridad incorporadas, como la prevención de ataques de inyección SQL.

Desventajas de ORM.

- **Tiempo utilizado en el aprendizaje.** Este tipo de herramientas suelen ser complejas por lo que su correcta utilización lleva un tiempo que hay que emplear en ver el funcionamiento correcto y ver todo el partido que se le puede sacar.
- Menor rendimiento (aplicaciones algo más lentas). Esto es debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá de transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.
- **Sistemas complejos.** Normalmente la utilidad de ORM desciende con la mayor complejidad del sistema relacional.

HERRAMIENTAS

Entre las herramientas ORM más relevantes encontramos las siguientes:

Hibernate:

Hibernate es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java (y disponible también para .Net con 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. Utiliza archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

Hibernate es software libre, distribuido bajo los términos de la licencia GNU LGPL.

Java Persistence Api (JPA):

El Java Persistence API (JPA) es una especificación de Sun Microsystems para la persistencia de objetos Java a cualquier base de datos relacional. Esta API fue desarrollada para la plataforma JEE e incluida en el estándar de EJB 3.0, formando parte de la Java Specification Request JSR 220.

Para su utilización, JPA requiere de J2SE 1.5 (también conocida como Java 5) o superior, ya que hace uso intensivo de las nuevas características de lenguaje Java, como las anotaciones y los genéricos.

iBatis:

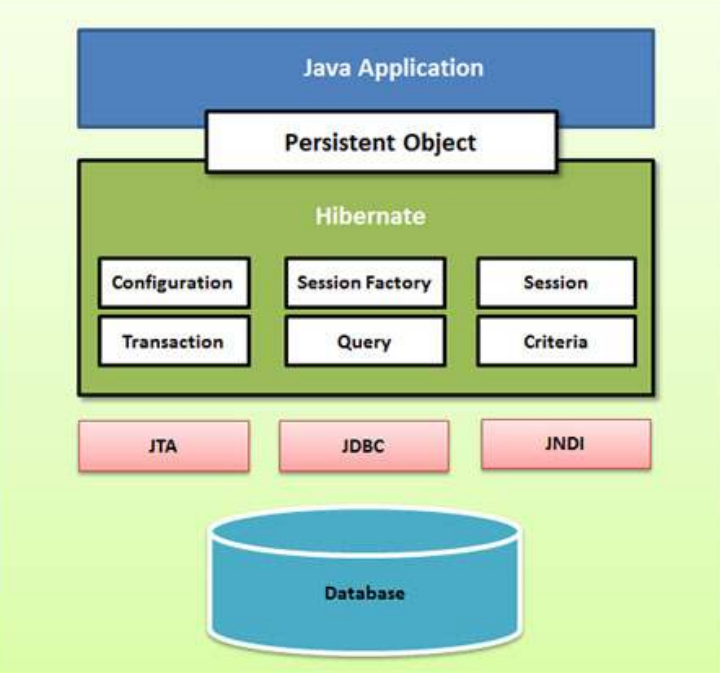
iBatis es un framework de persistencia desarrollado por la Apache software Foundation. Al igual que que el resto de los proyectos desarrollados por la ASF, iBatis es una herramienta de código libre.

iBatis sigue el mismo esquema de uso que Hibernate; se apoya en ficheros de mapeo XML para persistir la información contenida en los objetos en un repositorio relacional.

Arquitectura de Hibernate

La arquitectura de Hibernate es en capas para mantenerlo aislado de tener que conocer las API subyacentes. Hibernate usa los datos de la base de datos y proporciona persistencia entre los diferentes objetos. Los interfaces que presenta Hibernate son los siguientes:

Configuration: Representa un archivo de configuración o propiedades requerido por la hibernación. El objeto de configuración proporciona dos componentes claves: Conexión de bases de datos y configuración del mapeado de clase.
Session Factory: Configura Hibernate para la aplicación que utiliza el archivo de configuración suministrado y permite un objeto Session que se crea una instancia.
Session: Se utiliza para obtener una conexión física con una base de datos.
Transaction: Representa una unidad de trabajo con la base de datos y la mayor parte del RDBMS soporta la funcionalidad de transacción. Las transacciones en hibernación son manejadas por un gestor de transacciones subyacente y de transacción (de JDBC o JTA).
Query: Objetos de consulta SQL utilizan la cadena Hibernate Query Language (HQL) para recuperar datos de la base de datos y crear objetos.



Instalación y configuración de Hibernate

La instalación de Hibernate sobre el IDE NetBeans requiere tener instalado previamente este entorno de desarrollo, junto al JDK. El proceso que se va a desarrollar a continuación se hace con N**etBeans 7.2 aunque para la versión 8.2 el proceso es idéntico.**

El proceso de instalación de Hibernate se muestra en el siguiente documento que te puedes descargar en el siguiente enlace.

- Se destaca que el IDE NetBeans 8.X ya tiene incluida la versión 4.X de Hibernate, si deseas una versión superior debes descargarla según el enlace dado.

- Una vez que inicializamos NetBeans, podemos encontrar la herramienta Hibernate como un plugin de este entorno de desarrollo. Lo único que tienes que hacer es ir a la lista de plugins de NetBeans y comprobar que lo tienes instalado; en caso contrario, deberás instalarlo desde la opción de plugins disponibles.

- Captura de pantalla donde se muestra la lista de plugins instalados en NetBeans y, dentro de un rectángulo de borde rojo, se observa el plugin de Hibernate.
  Para utilizar Hibernate en una aplicación, es necesario conocer cómo configurarlo.

- Hibernate puede configurarse y ejecutarse en la mayoría de aplicaciones Java y entornos de desarrollo. El archivo de configuración de Hibernate recibe el nombre de Hibernate.cfg.xml y contiene información sobre la conexión de la base de datos y otras propiedades. Al crearlo, hay que especificar la conexión a la base de datos.

En NetBeans, cuando se crea el archivo de configuración de Hibernate usando el asistente, podemos especificar la conexión a la base de datos, eligiendo de una lista de conexiones de bases de datos registradas en el IDE. Cuando se genera el archivo de configuración, el IDE añade de forma automática detalles de la conexión e información basada en la conexión de la base de datos seleccionada. El IDE añade también las bibliotecas de Hibernate en el proyecto. Después de crear el fichero de configuración, éste puede ser editado usando el editor interactivo, o editar directamente el código XML.

Mapeo POJO - tabla BDR

Hibernate necesita saber dónde guardar cada dato de un objeto (POJO)
Un POJO (Plain Old Java Object) es una clase que...
- ... tiene un constructor por defecto
- ... tiene un atributo identificador (como la clave primaria de la BDR)
- ... los atributos deben ser privados, y tener un getter y un setter
- ... no es final*

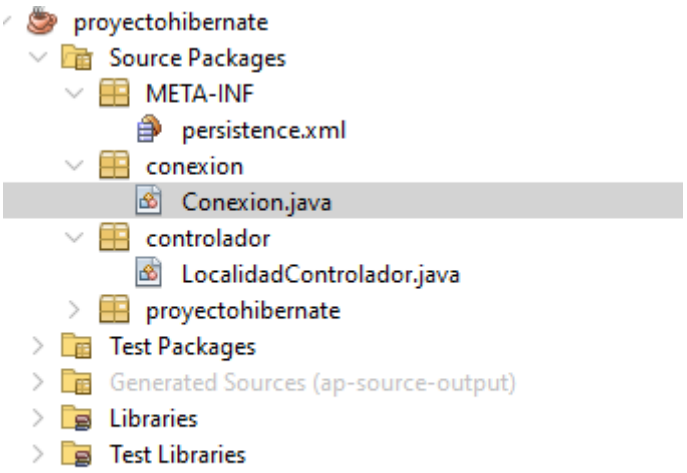
Cada POJO se corresponde con una entrada de una tabla

Ejemplo de JPA (Java Persistence API)

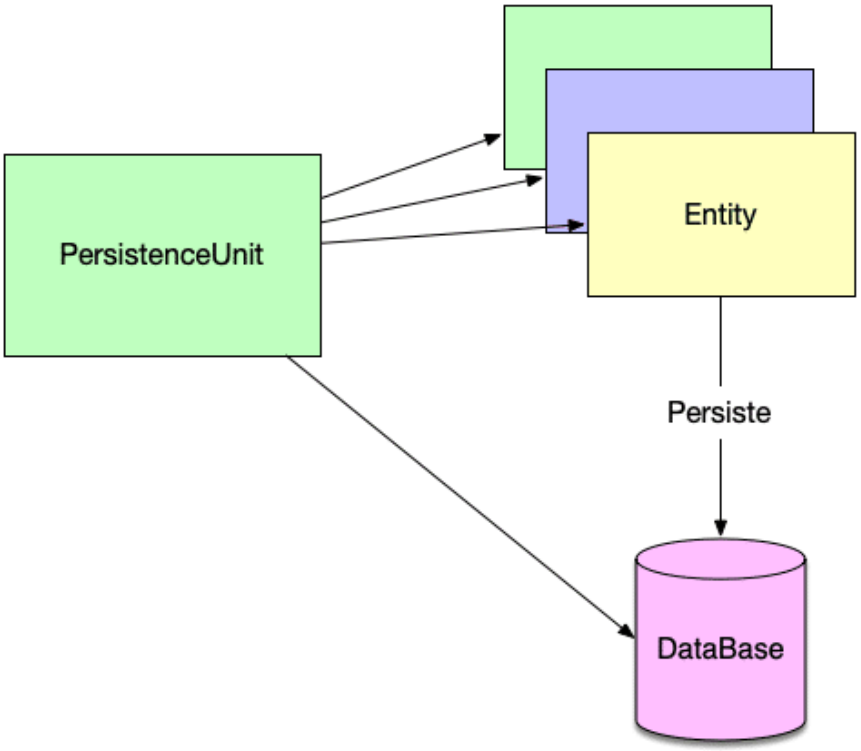
Java Persistence API y Persistence.xml

persistence.xml que se encuentra ubicado en la carpeta META-INF de un proyecto Java EE . Se encarga de conectarnos a la base de datos y define el conjunto de clases que vamos a gestionar. El fichero es parte del standard y existirá en cualquier implementación de JPA que se utilice.Contiene

- configuración de base de datos
- entidades de base de datos.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
             http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
             version="2.1">
  <persistence-unit name="proyectohibernatePU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>proyectohibernate.Empleado</class>
    <class>proyectohibernate.Pub</class>
    <class>proyectohibernate.Localidad</class>
    <class>proyectohibernate.Titular</class>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/pubs?zeroDateTimeBehavi
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.password" value=""/>
    </properties>
  </persistence-unit>
</persistence>
```



JPA y EntityManagerFactory

Por lo tanto hay que tener en cuenta que en Java Persistence API el fichero persistence.xml define la conexión a la base de datos y las entidades que vamos a

considerar salvables en ella. Como resultado de ambos conceptos se genera un objeto **EntityManagerFactory** que se encargará de guardar todas estas entidades para la base de datos.

Se tendrá una EntityManagerFactory con el que empezar a guardar las entidades que se encuentran definidas a nivel del fichero persistence.xml. Eso sí, muchas aplicaciones JEE conectan a varias bases de datos y tendrán diferentes EntityManagerFactorys . Cada uno estará ligado a un PersistenceUnit diferente.

Lo habitual es disponer de un único EntityManagerFactory con el que nosotros gestionamos todas las entidades. Por lo tanto cada fichero Persistence.xml, EntityManagerFactory y PersistenceUnit tiene sus propias responsabilidades.

//En java se crea de la siguiente manera:

EntityManagerFactory fabrica;
fabrica = Persistence.createEntityManagerFactory("proyectohibernatePU");

Manejo del EntityManager

Una vez tenemos de un EntityManagerFactory , este será capaz de construir un objeto EntityManager que facilita la persistencia de los objeto.

Estas entidades son objetos POJO (Plain Old Java Object) con los que estamos trabajando en nuestro programa . Por consiguiente , el EntityManager será el encargado de realizar todas las operaciones de tipo CRUD (insertar , borrar ,seleccionar y actualizar etc) sobre ellos.

EntityManager este será capaz de controlar todos los cambios que se han realizado en él y ejecutar las consultas adecuadas contra la base de datos

Ejemplo

```
mport javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import es.curso.bo.Persona;
public class Principal01Add {
   public static void main(String[] args) {
      Persona yo = new Persona("pedro", 25);
      EntityManagerFactory emf =
         Persistence.createEntityManagerFactory("UnidadPersonas");
```

```
  EntityManager em = emf.createEntityManager();
try {
    em.getTransaction().begin();
    em.persist(yo);
    em.getTransaction().commit();
 } catch (Exception e) {
```