# ACCESO-A-DATOS-TEMA-7.pdf

user_4383848

Acceso a datos

1° Desarrollo de Aplicaciones Multiplataforma

Estudios España

# ACCESO A DATOS - TEMA 7
# EL MAPEO OBJETO RELACIONAL

## 1. DEFINICIÓN:

Java es un lenguaje de programación orientado a objetos el cual puede ser representado en un gráfico de objetos; mientras que una base de datos relacional se representa en un formato tabular usando filas y columnas.
Cuando se trata de grabar un objeto en base de datos, existen algunas **diferencias obvias entre estos dos sistemas**; por lo tanto, para solventar este tipo de problemas o diferencias de datos a almacenar, tenemos la ayuda del **Mapeo Objeto Relacional**. Podemos definir un **ORM** como un **framework que facilita el almacenamiento de los objetos a una base de datos relacional.**
En cualquier base de datos, el nivel de elemento más alto es la tabla, dividida a su vez en filas y columnas. Una columna contiene valores en un tipo determinado, y una fila contendrá un conjunto de información de una tabla determinada. **Un objeto plano es equivalente a una fila en una base de datos relacional**, y básicamente esto es lo que se mapea en un sentido y en otro.

*Fig. 2. Esquema Mapeo*

## 2. VENTAJAS E INCONVENIENTES DEL MAPEO OBJETO RELACIONAL:

Java posee una API de conexión a base de datos: los drivers de conexión (JDBC) para acceder a la base de datos. El desarrollador debe conocer la base de datos a fondo, y saber qué tipo de relaciones existen entre tablas, así como el nombre exacto de las columnas, Constraints, etc.
Si consiguiéramos realizar las mismas operaciones con los datos desde la parte de Java, la perspectiva cambiaría totalmente. **De esta forma tendríamos abstracción, herencia, composición, identidad y muchas características más en la parte de la aplicación Java** que se podrían ***conseguir igualmente por medio de un framework*** (algunas de las razones por las que se usan los ORM).

Algunas de las **ventajas** de usar un ORM son:
• Eficiencia del desarrollo.
• Desarrollo más orientado a objetos.
• Manejabilidad.
• Facilidad para introducir nuevas funciones como el cacheo de información, por ejemplo.
Algunas de los **inconvenientes** que también hay que tener en cuenta son:
• Consume muchos recursos del sistema.
• La sintaxis de los ORM a veces puede complicarse si realizamos consultas muy complejas.

## 3. FASES DEL MAPEO OBJETO RELACIONAL:

**Fase 1:** en este punto **nos centramos en los datos del objeto**. Esta fase contiene los POJO (Plain Old Java Object), las clases simples de Java, las clases de implementación, clases e interfaces con su correspondiente capa de negocio de cada aplicativo (a esta capa la podemos llamar capa servicio y en ella también encontraremos las distintas clases DAO), además de clases orientadas a la capa de datos con métodos como crearObjeto(), encontrarObjeto(), borrarObjeto(), etc.

**Fase 2:** esta fase es llamada también de **persistencia o mapeo**. Contiene los siguientes agentes:
• Proveedor JPA: librería que hace posible toda la funcionalidad de JPA: javax.persistence.
• Archivo de asignación: es un fichero XML donde se almacena la configuración de la asignación de los datos de una clase JAVA (POJO) y los datos reales de la base de datos relacional.
• JPA Cargador: realmente esta parte funciona como una memoria caché, cargará los datos de la base de datos proporcionando algo parecido a una copia; para de esa forma, realizar interacciones rápidas con las clases de servicio.
• Reja de objeto: es el lugar donde se almacenan temporalmente una copia de los datos de nuestra base de datos relacional. Se le llama objeto grid, por lo que todas las consultas pasarán por este punto, y una vez realizadas las verificaciones pasarán a la base de datos principal.

**Fase 3:** llamada **fase de datos relacionales**. Una vez pasada la reja de objetos y todo haya ido bien, se irá directamente a base de datos. Hasta entonces como hemos mencionado antes, se permanecerá en ese espacio temporal de caché.

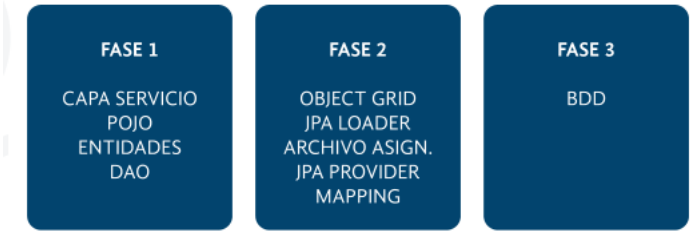

*Fig. 3. Esquema arquitectura ORM*

## 4. HERRAMIENTAS ORM:

**EBEAN:**
• Soporte en bases de datos: soporta bases de datos como H2, Postgres, Mysql, NuoDB, PostGis, MariaDB, SQL Server, Oracle, SAP, etc.
• Múltiples niveles de abstracción: consultas de tipo ORM mezcladas con SQL, además de consultas DTO.
• Beneficios: evita automáticamente N+1, usa caché de tipo L2 para reducir carga de base de datos, realiza consultas mezclando base de datos y cacheado L2, ajusta automáticamente las consultas ORM, contiene tecnología Elasticsearch para caché L3, etc.

**IBATIS:** En 2010 el desarrollo se centralizó en Google Code, usando el nuevo nombre **MyBatis**. Posee soporte para Java y .Net.
• Posee la opción de dividir la capa de persistencia en: capa de abstracción, capa de framework persistente, y capa de Driver.
• Una de sus virtudes es la facilidad de interactuar con los objetos y los datos de las bases de datos relacionales.
• Ofrece abstracción a nivel de la capa de persistencia de objetos.

**HIBERNATE: es el ORM más extendido y más usado**. Disponible para lenguaje **Java y también para .Net** (denominándose para éste, Nhibernate).
Facilita el mapeo relacional de los distintos objetos entre una base de datos relacional y el modelo de objetos de la aplicación; para lo que se apoya en un fichero .xml que representa y establece dichas relaciones o también por medio de anotaciones donde se establecen las relaciones.

**• Simplicidad:** al disponer de un solo fichero .xml para establecer las relaciones, es muy sencillo e intuitivo tanto dirigirnos a éste como consultar cualquier tipo de relación entre entidades o atributos.
**• Robusto:** En la capa de abstracción ofrece una propia capa de consultas SQL llamada HQL, orientada a facilitar la sintaxis y a mejorar la eficiencia de estas.

## 5. ARQUITECTURA DE HIBERNATE:

Incluye distintos objetos como son los **objetos de persistencia, sesiones, transacciones** etc. Hibernate usa el principio de reflexión Java, que permite el análisis y la modificación de los distintos atributos y características de las distintas clases en tiempo de ejecución.

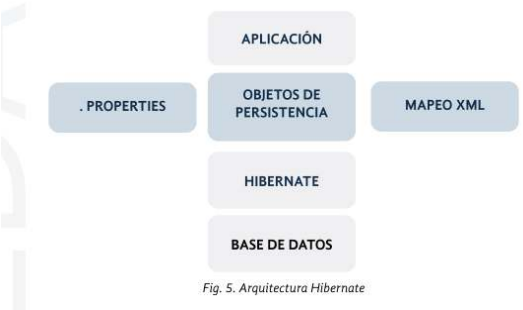

*Fig. 5. Arquitectura Hibernate*




## 6. COMPONENTES DE HIBERNATE:

• **SessionFactory Object:** mediante el que se permitirá el uso de objetos de tipo Session. Se puede instanciar de diferentes formas: normalmente coge la configuración existente en el fichero de configuración establecido por defecto. Utilizaremos un objeto SessionFactory por cada base de datos que tengamos en la aplicación.
• **Session Object:** Para instanciar una conexión directa con nuestra base de datos relacional. Su función principal es interactuar con la base de datos. No debe permanecer abiertos mucho tiempo por temas de seguridad.
• **Transaction Object:** es un objeto opcional, que básicamente maneja las transacciones directamente con las bases de datos relacionales. Si no se quiere hacer uso de dicho objeto, también podemos indicar manualmente aquellos bloques que queremos que sean transaccionales.
• **Query Object:** en Hibernate disponemos de varias formas de realizar consultas a la base de datos. Este tipo de objetos utilizan consultas de tipo SQL o de tipo Hibernate Query (HQL). Con este tipo de objetos enlazaremos los distintos parámetros de nuestra consulta, podremos realizar ciertas restricciones como controlar el número de resultados, y ejecutar la consulta. Es un modo de realizar consultas mucho más dinámico que las consultas nativas.
• **Criteria Object:** desaparecerá el lenguaje nativo de SQL para dar paso a las consultas por medio de objetos Java, y por medio de las funciones que nos ofrece Hibernate, que más tarde serán traducidas a sentencias SQL.

***Los puntos 9 y 10 del PDF son instalación y configuración de HIBERNATE.***