

## Mapeo objeto relacional.

### Caso práctico

En BK todo el desarrollo de software se hace siguiendo el paradigma de la Programación Orientada a Objetos. Sin embargo, la mayoría de las bases de datos con las que interaccionan sus aplicaciones son bases de datos relacionales, aunque también las hay orientadas a objetos.

**Ada** constata que parte de sus trabajadores se van a enfrentar con un dilema: ¿cómo usar objetos y clases para interactuar con bases de datos relacionales? Si en éstas sólo se pueden registrar tipos de datos simples, ¿cómo guardar ahí un objeto?

Este es el punto de partida del mapeo objeto-relacional, y los trabajadores de BK van a aprender cómo se trabaja en el sistema de almacenamiento de componentes propios de la Programación orientada a Objetos.



**Materiales formativos de F.P. Online propiedad del Ministerio de Educación, Cultura y Deporte.**

[Aviso Legal](#)

# 1.- Concepto de Mapeo objeto-relacional.

## Caso práctico

**María** percibe que sus aplicaciones, basadas en Programación Orientada a Objetos, no son almacenables de forma directa en los Sistemas Gestores de Bases de Datos que tienen en la empresa. ¿Cómo guardar un objeto en un sistema de almacenamiento basado en tablas?



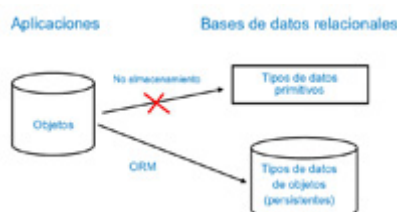
A la hora de almacenar los datos de un programa orientado a objetos en una [base de datos relacional](#), surge un inconveniente debido a incompatibilidad de sistemas de [tipos de datos](#). En el software orientado a objetos, la información se representa como clases y objetos. En las bases de datos relacionales, como tablas y sus restricciones. Por tanto, para almacenar la información tratada en un programa orientado a objetos en una base de datos relacional es necesaria una traducción entre ambas formas.

El mapeo objeto-relacional (ORM) soluciona este problema. 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](#)).

**La mayoría de las aplicaciones se construyen usando técnicas de programación orientada a objetos; sin embargo, los sistemas de bases de datos más extendidos son de tipo relacional.**

Las bases de datos más extendidas son del tipo relacional y estas 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.



El ORM se encarga, de forma automática, de convertir los [objetos](#) en [registros](#) y viceversa, simulando así tener una base de datos orientada a objetos.

## Para saber más

En el siguiente enlace encontrarás un documento muy interesante sobre la problemática asociada al almacenamiento de objetos y al concepto de mapeo objeto-relacional.

[Introducción al mapeo objeto-relacional.](#) (0.43 MB)

## 2.- Herramientas ORM. Características y herramientas más utilizadas.

### Caso práctico

En la empresa se ponen manos a la obra para estudiar las herramientas que permitan almacenar objetos y clases en bases de datos relacionales. ¿Qué son y qué características tienen las herramientas ORM?



Ya hemos dicho que las herramientas ORM se utilizan para dar solución al problema de que, en la programación orientada a objetos, la gestión de datos se implementa usando objetos; sin embargo, en los sistemas de gestión de bases de datos SQL sólo se pueden almacenar y manipular valores escalares organizados en tablas relacionales.

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 escalares, la 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.**

Las herramientas ORM pues, actúan como un puente que conecta las ventajas de los RDBMS con la buena representación de estos en un lenguaje Orientado a Objetos, o, dicho en otras palabras, nos lleva de la base de datos al lenguaje de programación.

### Para saber más

Qué son y cómo actúan las herramientas ORM. En el siguiente enlace encontrarás un artículo muy interesante sobre los tipos de herramientas ORM más utilizados en la actualidad y sus características más importantes.

[Tipos y características de las herramientas ORM.](#)

En el siguiente documento podrás ver un adelanto de los siguientes puntos de la unidad:

[Resumen textual alternativo](#)

## Autoevaluación

Utilizando una herramienta ORM podemos recuperar los objetos de un programa en formas compatibles con el sistema de almacenamiento de los gestores de bases de datos relacionales.

- ☐ No.
- ☐ Si.



## 2.1.- Características.

### Caso práctico

**Antonio** no sabe exactamente qué tiene que utilizar ni qué ventajas obtendrá con el uso de las herramientas de mapeo ¿Cuáles son las características de las herramientas ORM?



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. Gracias a las ORM, 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 POJO's 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 utilizando VO's directamente.

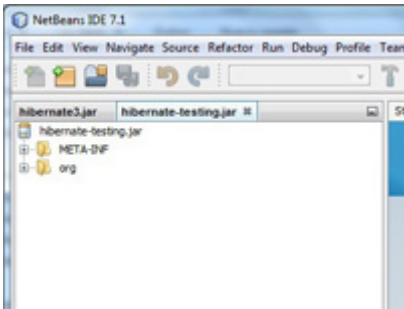
**Una herramienta ORM permite tomar un objeto Java y hacerlo persistente, carga el objeto de la base de datos a memoria y permite hacer consultas a las tablas de la base de datos.**

#### 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.
- ✓ Abstracción de la base de datos.
- ✓ Reutilización.
- ✓ Permiten persistir objetos a través de un método `Orm.Save` y generar el SQL correspondiente.
- ✓ Permiten recuperar los objetos persistentes a través de un método `Orm.Load`.
- ✓ Lenguaje propio para realizar las consultas.
- ✓ Independencia de la base de datos.
- ✓ Incentivan la [portabilidad](#) y [escalabilidad](#) de los programas de software.

#### 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 descende con la mayor complejidad del sistema relacional.



## 2.2.- Herramientas ORM más utilizadas.

### Caso práctico

Sabiendo ya la función de las herramientas ORM, el siguiente punto es el análisis de las ORM existentes en el mercado, junto a sus características más destacables, para decidir cuál es la que mejor se adapta a nuestras necesidades.



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 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.



En el siguiente documento encontrarás las características más destacables de la herramienta Hibernate.

[Aspectos de la configuración de Hibernate.](#) (0.19 MB)



### Autoevaluación



**La herramienta ORM Hibernate se caracteriza por:**

- ☐ No ser compatible con la plataforma .Net.
- ☐ Ser software libre.
- ☐ Ser una especificación de Sun Microsystems.

## 3.- Instalación y configuración de Hibernate

### Caso práctico

Una vez que el equipo de BK ha estudiado las diferentes herramientas ORM, ya han decidido que es Hibernate la que mejor se adapta a sus requerimientos. Por ello, el paso siguiente es su instalación y configuración en los equipos para empezar a utilizarla.



La instalación de Hibernate sobre el IDE NetBeans requiere tener instalado previamente este entorno de desarrollo, junto al JDK.

### Debes conocer

La instalación del JDK y del entorno de desarrollo NetBeans en Linux. En el siguiente enlace accederás a una animación para que sepas cómo realizar la instalación de ambos.

[Resumen textual alternativo](#)

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.

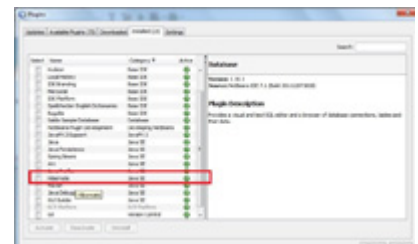
## Debes conocer

La instalación de Hibernate en NetBeans. En la siguiente animación puedes ver cómo se hace.

[Resumen textual alternativo](#)

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 los siguientes puntos del tema veremos más detenidamente el proceso de configuración.



## Autoevaluación

### Ejercicio de relacionar

Tipo de herramienta	Relación	Desarrollador
Hibernate.	<input type="checkbox"/>	1. Sun Microsystems.
openJPA.	<input type="checkbox"/>	2. Apache software Foundation.
iBatis.	<input type="checkbox"/>	3. Oracle.

Enviar

## 4.- Ficheros de configuración y mapeo. Estructura y propiedades.

### Caso práctico

En el momento de empezar a trabajar, **Juan** se enfrenta a aspectos esenciales tras la instalación de la herramienta ORM: los ficheros de configuración y de mapeo para el correcto funcionamiento de Hibernate.



Para empezar a trabajar con Hibernate es necesario configurar la herramienta para que conozca qué objetos debe recuperar de la base de datos relacional y en qué lugar los hará persistir. Por tanto, el primer paso será tener una base de datos relacional con la que poder trabajar.

### Debes conocer

La creación y conexión de la base de datos relacional, sakila, con la que trabajará Hibernate. En la siguiente animación puedes ver cómo usar Sakila, una base de datos de ejemplo.

[Resumen textual alternativo](#)

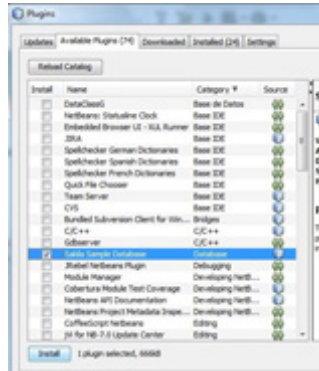
En NetBeans, cuando se crea el archivo de configuración de Hibernate usando el asistente, podemos especificar la conexión a la base de datos, eligiendo de una lista de conexiones de bases de datos registradas en el IDE. Cuando se genera el archivo de configuración, el IDE añade de forma automática detalles 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.

El fichero de configuración contiene información sobre la base de datos a la que vamos a conectar la aplicación. Si se la aplicación se va a conectar a varias bases de datos, sería necesario definir tantos archivos de configuración como bases de datos a las que nos queramos conectar.

Para tener toda esta información, en Hibernate surgen dos ficheros distintos:

- ✓ El archivo de propiedades de Hibernate (`Hibernate.properties`), que es el encargado de determinar todos los aspectos relacionados con el gestor de la base de datos y las conexiones con él.
- ✓ Los archivos que definen el emparejamiento (mapping) de propiedades con tablas y columnas (`*.hbm.xml`).



**Para utilizar Hibernate necesitamos tener una base de datos relacional instalada en NetBeans. La base de datos Sakila es una muestra gratuita disponible, como plugin, en NetBeans.**

**La seleccionamos desde la lista de plugins disponibles y la instalamos en nuestro IDE.**

Como ya se ha visto, tras instalar el plugin podemos crear la base de datos Sakila desde la ventana Servicios de NetBeans. Para ello, seleccionaremos el servidor de MySQL y a continuación esta base de datos. Conectaremos con ella para poder tenerla disponible cuando empecemos a extraer la información que se nos pida.

## Para saber más

En el siguiente enlace encontrarás un documento donde se expone con más detalle aspectos básicos de configuración de Hibernate.

[Tutorial de Hibernate.](#)

## 4.1.- Ficheros de configuración. Propiedades

### Caso práctico

**Ana** ya ha instalado Hibernate sobre el IDE NetBeans y ahora surge el problema de configurar correctamente todos sus parámetros para trabajar de la forma más adecuada. Hay varias formas de trabajar con los archivos de configuración de Hibernate; habrá que decidir de qué forma se va a trabajar.



El archivo de configuración de Hibernate es el **Hibernate.cfg.xml** y contiene información sobre la conexión de base de datos, las asignaciones de recursos y otras propiedades de conexión.

### Debes conocer

El proceso de creación de una aplicación Java en NetBeans y la configuración de Hibernate para poder acceder posteriormente a la base de datos sakila.

[Resumen textual alternativo](#)

Al seleccionar una conexión a una base de datos en NetBeans, se crea un archivo de configuración en Hibernate que guarda los detalles de esa conexión. NetBeans también añade de forma automática la biblioteca de Hibernate para la ruta de clase que tendrá nuestro proyecto. Después de la creación del archivo **Hibernate.cfg.xml**, podemos editarlo o modificar el código xml mediante el editor xml.



Las propiedades más importantes del fichero **Hibernate.cfg.xml** son:

- ✓ `Hibernate.dialect`: Dialecto o lenguaje empleado. Por ejemplo, MySQL.
- ✓ `Hibernate.connection.driver_class`: Driver utilizado para la conexión con la base de datos.
- ✓ `Hibernate.connection.url`: Dirección de la base de datos con la que se va a conectar Hibernate.
- ✓ `Hibernate.connection.username`: Nombre del usuario que va a realizar la extracción de información. Por defecto, el nombre de usuario es root.
- ✓ `Hibernate.connection.password`: Contraseña el root.
- ✓ `Hibernate.show_sql`: Para mostrar la herramienta. Por defecto, su valor es true.



## Autoevaluación

El archivo `Hibernate.cfg.xml` contiene información sobre:

- ☐ La conexión, recursos y otras propiedades como nombre de usuario, contraseña, etc.
- ☐ La conexión a la base de datos.
- ☐ La dirección de la base de datos y la versión del IDE utilizada.



## 4.2.- Ficheros de mapeo. Estructura, elementos y propiedades.

### Caso práctico

Ya se ha configurado Hibernate y todo parece estar listo. La cuestión ahora es relacionar los objetos del programa orientado a objetos con las tablas de la base de datos relacional. Entran en juego los ficheros de mapeo.



Hibernate utiliza ficheros de mapeo para relacionar tablas con objetos Java, estos ficheros están en formato XML y que tienen extensión **.hbm.xml**.

**Hibernate-mapping:** Todos los ficheros de mapeo comienzan y acaban con esta etiqueta. Al comienzo indicamos en el atributo **package** el paquete Java donde se encuentra la clase.

Los archivos de mapeo indican la correspondencia entre el bean y una tabla de la base de datos; por tanto, en el momento en que se inicie una tarea que requiera el acceso a la base de datos se obtendrá una conexión **JDBC**.

A los archivos de mapeo se les asigna el nombre de la Clase seguido de la extensión **"hbm.xml"**.

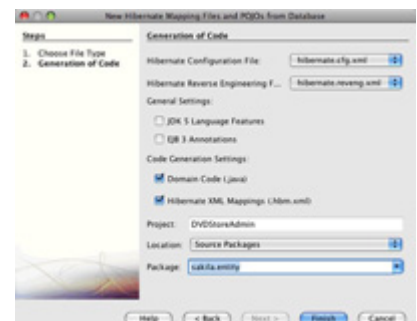
Podemos utilizar la ingeniería inversa para crear archivos de mapeo basados en tablas de la base de datos que seleccionemos. El archivo de ingeniería inversa es **Hibernate.reveng.xml**. Se trata de un archivo XML que se puede utilizar para modificar la configuración predeterminada de **Hibernate.cfg.xml** con el propósito de especificar explícitamente el esquema de base de datos que se va a utilizar, filtrar las tablas que no deseamos recuperar y especificar cómo se asignan los tipos JDBC a los tipos de Hibernate.

**Para extraer una tabla concreta de la base de datos, la sintaxis en el mapeo requiere definir el POJO Nombre\_Clase.hbm.xml, donde nombre\_clase se corresponderá con el nombre la tabla que queremos extraer y donde se describe cómo se relacionan clases y tablas y propiedades y columnas.**

Mediante el asistente de NetBeans, seleccionamos Nuevo Mapeo en Hibernate y rellenamos los campos que nos piden, en función de la información que nos interese extraer de la base de datos.

De esta manera, NetBeans genera un POJO **nombre\_clase.java** (siendo **nombre\_clase** aquella tabla cuya información nos interesa extraer de la base de datos) con todos los campos necesarios.

Con ello, genera un fichero de mapeo de Hibernate, agregando su entrada a **Hibernate.cfg.xml**, entre las etiquetas .



### Debes conocer

El proceso de creación de archivos de mapeo en Hibernate.



[Resumen textual alternativo](#)



## Autoevaluación

Los ficheros de mapeo se utilizan para relacionar tablas y clases que deseamos extraer de la base de datos que estamos consultando.

- ☐ Falso.
- ☐ Verdadero.



## 5.- Mapeo de colecciones, relaciones y herencia-

### Caso práctico

**María** necesita ahora aprender la verdadera utilidad de Hibernate: el mapeo de elementos, tales como relaciones, colecciones y herencia, que le permita avanzar sobre el objetivo perseguido de almacenar estas características de la Programación Orientada a Objetos en las bases de datos relacionales.



#### 1. Mapeo de colecciones.

Hay bastantes rangos de mapeos que se pueden generar para colecciones que cubran diversos modelos relacionales.

El elemento de mapeo de Hibernate utilizado para mapear una colección depende del tipo de la interfaz; por ejemplo, un elemento se utiliza para mapear propiedades de tipo Set aunque, no obstante, existen además los elementos de mapeo etc.



Las instancias de colección se distinguen por la clave foránea de la entidad que posee la colección. El elemento mapea la columna clave de la colección.

Hay que resaltar que las colecciones pueden contener casi cualquier tipo de datos; esto implica que un objeto en una colección puede ser manejado con una semántica de "valor" o podría ser una referencia a otra entidad, con su propio ciclo de vida. Todos los mapeos de colección necesitan una columna índice en la tabla de colección: una columna índice es una columna que mapea a un índice de array o índice de List. Por último, una colección de valores o asociación muchos-a-muchos requiere una tabla de colección dedicada con una columna o columnas de clave foránea, columna de elemento de colección y probablemente una o varias columnas índice.

#### 2. Mapeo de relaciones.

Para persistir, las relaciones usan las denominadas transacciones, ya que los cambios pueden incluir varias tablas. Una regla general para el mapeo es respetar el tipo de relación en el modelo de objetos y en el modelo relacional: así, una relación 1-1 en el modelo de objetos deberá corresponderse a una relación 1-1 en el modelo relacional.

Para mapear las relaciones, se usan los identificadores de objetos (OID). Son la llave primaria de la tabla relacionada y se agregan como una columna más en la tabla donde se quiere establecer la relación. Dicha columna es una clave foránea a la tabla con la que está relacionada.

#### 3. Mapeo de herencia.

Como hemos comentado, las colecciones funcionan en ambos modelos, objeto y relacional. Para el caso de la herencia se presenta el problema que las bases de datos relacionales no la soportan. Así es que somos nosotros quienes debemos modelar como se verá la herencia en el modelo relacional. Existen tres tipos de mapeos principales: modelar la jerarquía a una sola tabla, modelar la jerarquía completa en tablas, mapear cada tabla en tablas concretas. La decisión estará basada en el rendimiento y en la escalabilidad del modelo.

### Para saber más

En el siguiente enlace puedes tener acceso a un tutorial en línea de los mapeos en Hibernate, donde se detallan estos aspectos con más profundidad.

[Mapeos en Hibernate.](#)

## 6.- Clases persistentes.

### Caso práctico

Para poder interactuar con un bases de datos, **María** debe definir en su aplicación clases y objetos que puedan representar las entidades de la base de datos con la que interacciona la aplicación.



Se denomina "persistencia" de los objetos a su capacidad para guardarse y recuperarse desde un medio de almacenamiento. Las clases persistente son clases en una aplicación que nos van a servir para representar entidades de la base de datos. No se considera que todas las instancia de una clase persistente puedan estar en estado persistente, ya que una instancia puede ser transitorio o separada.

El estándar Java Data Objects (JDO), define una clase con capacidad de persistencia, como aquella que implementa la interface `javax.jdo.PersistenceCapable`.



Las clases persistentes tienen la capacidad de definir objetos que pueden almacenarse y recuperarse y un almacén persistente de datos. La especificación JDO incorpora la figura del procesador de clases en código ejecutable Java, JDO Enhacer, que es un programa que modifica los archivos compilados de las clases, añadiendo el código ejecutable necesario para realizar la grabación y recuperación transparente de los atributos de las instancias persistentes.

JDO permite a los programadores y programadoras convertir sus clases en persistentes, de forma que los objetos pertenecientes a clases concretas definidas por el programador pueden mantener su estado, con la única limitación de que el estado esté compuesto por los atributos persistentes que sean independientes del contexto de ejecución: tipos primitivos, tipos de referencia e interfaz y algunas clases del sistema que permiten modelar el estado como por ejemplo la clase `Array`, `Date`, etc.

Para poder indicar las clases y atributos que son persistentes, se utiliza un fichero de configuración XML, que se denomina descriptor de persistencia.. Para que las instancias de las clases persistentes pueden mantenerse en los sistemas gestores de bases de datos, es necesario establecer la correspondencia entre los objetos y su estado persistente.

### Para saber más

El siguiente enlace nos muestra documentación sobre Java Data Objects. (En inglés)

[Java Data Objects.](#)

## 7.- Sesiones; estados de un objeto.

### Caso práctico

**Ana** se pregunta si los objetos que se utilizan en una aplicación, para representar entidades de una base de datos, se mantienen siempre en el mismo estado, o si por el contrario, durante todo el proceso de inicio de aplicación, acceso a la base de datos, modificación, etc., estos objetos cambian de estado o funcionalidad.



Para poder utilizar la persistencia en Hibernate es necesario definir un objeto **Session** utilizando la clase **SessionFactory**. La sesión corresponde con un objeto que representa una unidad de trabajo con la base de datos. La sesión nos permite representar el gestor de persistencia, ya que dispone de una API básica que nos permite cargar y guardar objetos.



La sesión está formada internamente por una cola de sentencias SQL que son necesarias ejecutar para poder sincronizar el estado de la sesión con la base de datos.

Asimismo, la **sesión** contiene una lista de objetos persistentes. Una sesión corresponde con el primer nivel de caché.

Si para realizar el acceso a datos, usamos Hibernate, la sesión nos permite definir el alcance de un contexto determinado. Para poder utilizar los mecanismos de persistencia de Hibernate se debe inicializar el entorno Hibernate y obtener un objeto **Session** utilizando la clase **SessionFactory** de Hibernate. Un objeto **Session** Hibernate representa una única unidad-de-trabajo para un almacén de datos dado y lo abre un ejemplar de **SessionFactory**. Se deben cerrar las sesiones cuando se haya completado todo el trabajo de una transacción.

Los estados en los que se puede encontrar un objeto son:

- ✓ **Transitorio (Transient).** En este estado estará un objeto recién creado que no ha sido enlazado con el gestor de persistencia.
- ✓ **Persistente:** En este caso el objeto está enlazado con la sesión. Todos los cambios que se realicen serán persistentes.
- ✓ **Disociado (Detached):** En este caso nos encontramos con un objeto persistente que sigue en memoria después de que termine la sesión. En este caso existe en Java y en la base de datos.
- ✓ **Borrado (Removed):** En este caso el objeto está marcado para ser borrado de la base de datos. Existe en la aplicación Java y se borrará de la base de datos al terminar la sesión.



### Autoevaluación

**Un objeto en estado transitorio:**

- ☐ El objeto no ha sido enlazado con el gestor de persistencia.
- ☐ El objeto ha sido marcado para ser borrado.
- ☐ El objeto sigue en memoria después de finalizada la sesión.



## 8.- Carga, almacenamiento y modificación de objetos.

### Caso práctico

**Juan** se está convirtiendo en un entusiasta de las herramientas de mapeo-objeto relacional. Empieza a entender y comprobar sus ventajas, sin embargo, ahora toca lo más importante: como cargar, almacenar y modificar los objetos que representan elementos de la base de datos.



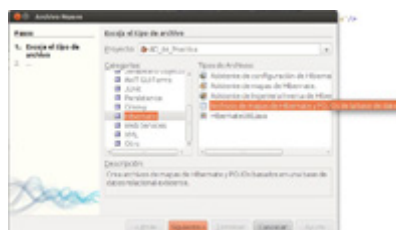
Para cargar un objeto de acceso a datos en la aplicación Java, el método `load()` de la clase `Session` suministra un mecanismo para capturar una instancia persistente, si conocemos su identificador. El método `load()` acepta un objeto `Class`, y cargará el estado de una nueva instancia de esa clase, inicializada en estado persistente.

El método `load()` lanzará una excepción irre recuperable si no existe la fila de base de datos correspondiente. Si no se está seguro de que exista una fila correspondiente, debe usarse el método `get()`, el cual consulta la base de datos inmediatamente y devuelve `null` si no existe una fila correspondiente.

Existen dos métodos que se encargan de recuperar un objeto persistente por identificador: `load()` y `get()`. La diferencia entre ellos radica en cómo indican que un objeto no se encuentra en la base de datos: `get()` devuelve un nulo y `load()` lanza una excepción `ObjectNotFoundException`.

Aparte de esta diferencia, `load()` intenta devolver un objeto `proxy` siempre y cuando le sea posible (no esté en el contexto de persistencia). Con lo que es posible que la excepción sea lanzada cuando se inicialice el objeto proxy. Esto es conocido como carga perezosa.

En el caso que se obtenga un proxy, no tiene impacto sobre la base de datos (no se ejecuta ninguna consulta), hasta que no se inicializa el mismo. Muy útil cuando se obtiene una referencia de un objeto para asociarlo a otro. (No es necesario obtener el objeto). Se modifica un objeto persistente.



### Para saber más

El siguiente vídeo nos muestra el estado de objetos en Hibernate.

[Resumen textual alternativo](#)

## 8.1.- Almacenamiento y modificando de objetos persistentes.

### Caso práctico

**Juan** empieza a comprender como se pueden cargar objetos persistente. **Juan** quiere enseñar a a **María** el mecanismo de almacenamiento y modificación de objetos persistentes, para que la aplicación que enlaza a bases de datos sea lo más completa posible.



Para almacenar objetos persistentes se proceso siguiendo los siguientes pasos:

1. Se instancia un objeto nuevo (estado transitorio).
2. Se obtiene una sesión y se comienza la transacción, inicializando el contexto de persistencia.
3. Una vez obtenida da la sesión, se llama al método `save()`, el cual introduce el objeto en el contexto de persistencia. Este método devuelve el identificador del objeto persistido.
4. Para que los cambios sean sincronizados en las bases de datos, es necesario realizar el `commit` de la transacción. Dentro del objeto sesión se llama al método `flush()`. Es posible llamarlo explícitamente. En este momento, se obtiene la conexión JDBC a la bases de datos para poder ejecutar la oportuna sentencia.
5. Finalmente, la sesión se cierra, con el objeto de liberar el contexto de persistencia, y por tanto, devolver la referencia del objeto creado al estado disociado.

Los objetos cargados, grabados, creados o consultados por las sesión pueden ser manipulados por la aplicación, y cualquier cambio a su estado de persistencia será persistido cuando se le aplique "flush" a la sesión.

No hay que invocar ningún método en particular para que las modificaciones se vuelvan persistentes. Así que la manera más sencilla y directa de actualizar el estado de un objeto es cargarlo con `load()`, y luego manipularlo directamente, mientras la sesión esté abierta.

Para borrar objetos persistentes, podemos ejecutar `Session.delete()`, que quitará el estado de un objeto de la base de datos. Por supuesto, su aplicación podría aún contener una referencia al objeto quitado. Se puede borrar objetos en cualquier orden, no se van producir violaciones de llave externa, pero sí es posible violar **constraints NOT NULL** aplicadas a la columna de llave externa.

Muchas aplicaciones necesitan capturar un objeto en una transacción, mandarlo a la capa de interfaz de usuario para su manipulación, y grabar sus cambios en una nueva transacción. Las aplicaciones que usan este tipo de estrategia en entornos de alta concurrencia, normalmente usan datos versionados para garantizar aislamiento durante la "larga" unidad de trabajo.

Hibernate soporta este modelo, proveyendo "reasociación" de entidades desprendidas usando los métodos `Session.update()` o `Session.merge()`.







## Autoevaluación

Para sincronizar los datos con la base de datos es necesario realizar:

- ☐ Un inicio de sesión.
- ☐ Se invoca el método `save()`.
- ☐ Se invoca al método `flush()`.



## 9.- Consultas SQL.

### Caso práctico

**Ana** ya ha aprendido como poder utilizar una base de datos relacional en una aplicación que utiliza clases persistentes. La pregunta que le surge ahora es ¿Cómo puedo realizar las consultas SQL con esta forma de acceder a los datos? ¿Puedo utilizar el select, insert, etc.?



Usando Hibernate, la ejecución de consultas SQL nativas se controla por medio de la interfaz `SQLQuery`, la cual se obtiene llamando a `Session.createQuery()`. Las siguientes secciones describen cómo utilizar esta API para consultas

La consulta SQL más básica es para obtener a una lista de escalares (valores).

```
✓ sess.createQuery("SELECT * FROM Personas").list();
✓ sess.createQuery("SELECT ID,NOMBRE, EDAD FROM PERSONAS").list();
```



Estas retornarán una lista de objetos arrays (`Object[]`) con valores escalares para cada columna en la tabla `PERSONAS`. Hibernate utilizará `ResultSetMetadata` para deducir el orden real y los tipos de los valores escalares retornados.

Otro tipo de consulta más compleja, es la consulta de entidades. Para obtener los objetos entidades desde una consulta sql nativa, se utiliza por medio de `addEntity()`.

```
✓ sess.createQuery("SELECT * FROM PERSONAS").addEntity(Persona.class);
✓ sess.createQuery("SELECT ID,NOMBRE,EDAD FROM PERSONAS").addEntity(Persona.class);
```

Se especifica esta consulta:

- ✓ La cadena de consulta SQL.
- ✓ La entidad devuelta por la consulta.

Asumiendo que `Persona` es mapeado como una clase con las columnas `IDENTIFICACION`, `NOMBRE` y `FECHA DE NACIMIENTO` las consultas anteriores devolverán una Lista en donde cada elemento es una entidad `Persona`.



### Autoevaluación

¿En qué estado se encuentra un objeto persistente en memoria, una vez finalizada la sesión?

- ☐ Transitorio.
- ☐ Borrado.
- ☐ Disociado.



## 10.- Lenguajes propios de la herramienta ORM.

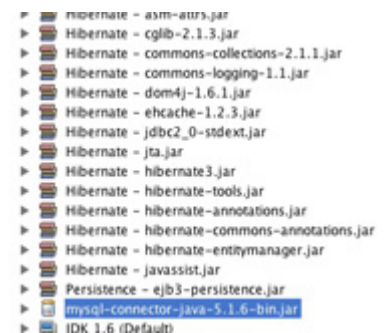
### Caso práctico

**María** conoce el lenguaje SQL, para poder realizar consultas a una base de datos relacional. Sin embargo están para poder interactuar con la base de datos desde Java, en BK está usando Hibernate. ¿Dispondrá Hibernate de un lenguaje de consulta propio? Ada le comenta que existe el HQL, que le va a facilitar el trabajo de consulta.



Hibernate utiliza un lenguaje de consulta potente (HQL) que se parece a SQL. Sin embargo, comparado con SQL, HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación. Las consultas se escriben en HQL y Hibernate se encarga de convertirlas al SQL usado por la base de datos con la que estemos trabajando y ejecutarla para realizar la operación indicada.

HQL es que es case-insensitive, o sea que sus sentencias pueden escribirse en mayúsculas y minúsculas. Por lo tanto "SeLeCt", "seleCt", "select", y "SELECT" se entienden como la misma cosa. Lo único con lo que debemos tener cuidado es con los nombres de las clases que estamos recuperando y con sus propiedades, ahí sí se distinguen mayúsculas y minúsculas. O sea, en este caso "pruebas.Hibernate.Usuario" NO ES LO MISMO que "PrueBAs.Hibernate.UsuArio".



Entre las características más importantes de HQL.

- ✓ Soporte completo para operaciones relacionales: HQL permite representar consultas SQL en forma de objetos. HQL usa clases y atributos o propiedades en vez de tablas y columnas.
- ✓ Regresa sus resultados en forma de objetos: Las consultas realizadas usando HQL regresan los resultados de las mismas en la forma de objetos o listas de objetos, que son más fáciles de usar.
- ✓ Consultas Polimórficas: Podemos declarar el resultado usando el tipo de la superclase e Hibernate se encargara de crear los objetos adecuados de las subclases correctas de forma automática.
- ✓ Soporte para características avanzadas: HQL contiene muchas características avanzadas que son muy útiles y que no siempre están presentes en todas las bases de datos, o no es fácil usarlas, como paginación, fetch joins con perfiles dinámicos, inner y outer joins, etc. Además soporta proyecciones, funciones de agregación (max, avg), y agrupamientos, ordenamientos, y subconsultas.
- ✓ Independiente del manejador de base de datos: Las consultas escritas en HQL son independientes de la base de datos (siempre que la base de datos soporte la característica que estamos intentando utilizar ^



### Autoevaluación

El lenguaje de consulta propio de Hibernate es:

- ☐ HQL.
- ☐ SQL.
- ☐ Java.



## 10.1.- Lenguaje HQL.

### Caso práctico

Para poder sacarle el máximo partido a las herramienta Hibernate, **Ada** es consciente que el equipo de BK debe conocer la sintaxis básicas del lenguaje HQL. Conociendo la forma de construir consultas con HQL, se abre un abanico muy grande de posibilidades, para la implementación de consultas de ciertas complejidad.



**Clausula from:** La consulta más simple que se puede realizar con Hibernate, es utilizando la cláusula **from**, la siguientes sería una consulta que mostraría todos los datos de una tabla de nombre Alumnos: **from Alumnos**

**Cláusula select:** La cláusula select escoge qué objetos y propiedades devolver en el conjunto de resultados de la consulta. Un ejemplo de consulta podría ser **select alumno.nombre from Alumnos alumno where alumno.nombre like 'A%'**

**La cláusula where:** La cláusula **where** nos permite refinar la lista de instancias retornadas. Si no existe ningún alias, puede referirse a las propiedades por nombre: **from Alumnos where nombre='Francisco'**. Si existe un alias, usaremos un nombre de propiedad calificado: **from Alumnos as alumnos where alumnos.nombre='Francisco'**. Esto retorna instancias de Alumnos llamados "Francisco".

**Funciones de agregación.** Las consultas HQL pueden retornar resultados de funciones de agregación sobre propiedades: **select avg(alumnos.nota), sum(alumnos.nota), max(alumnos.nota), count(alumnos) from Alumnos alumnos.**

**Expresiones.** Las expresiones utilizadas en la cláusula where incluyen lo siguiente: operadores matemáticos, operadores de comparación binarios, operadores lógicos , paréntesis ( ) que indican agrupación, funciones Java, etc.

**La cláusula order by.** La lista retornada por una consulta se puede ordenar por cualquier propiedad de una clase retornada o componentes. La palabra **asc** o **desc** opcionales indican ordenamiento ascendente o descendente respectivamente.

**La cláusula group by.** Una consulta que retorna valores agregados se puede agrupar por cualquier propiedad de una clase retornada o componentes:

**Subconsultas.** Para bases de datos que soportan subconsultas, Hibernate soporta subconsultas dentro de consultas. Una subconsulta se debe encerrar entre paréntesis (frecuentemente por una llamada a una función de agregación SQL). Incluso se permiten subconsultas correlacionadas (subconsultas que se refieren a un alias en la consulta exterior).

Sesión: hibernate.cfg  
from Actor|

FirstNa...	LastNa...	Actorid	LastUp...
PENELO	GUINNESS	1	2006-02...
NICK	WAHLBE	2	2006-02...
ED	CHASE	3	2006-02...
JENNIFER	DAVIS	4	2006-02...
JOHNNY	LOLOB	5	2006-02...
BETTE	NICHOL	6	2006-02...

### Debes conocer

El proceso de creación de consultas HQL en formularios con Hibernate en NetBeans para extraer información almacenada en una base de datos relacional.

[Resumen textual alternativo](#)

## Para saber más

En la siguiente dirección Web, encontrarás una amplia información sobre el lenguaje de consulta HQL.

[Tutorial de HQL.](#)

## 11.- Gestión de transacciones.

### Caso práctico

**Juan** está muy contento con la implementación de la interacción de la aplicación con la base de datos. El uso de clases y objetos que de forma transparente realizan las operaciones con las base de datos, supone un gran ventaja en la programación de aplicaciones con acceso a bases de datos. Sin embargo le surge una duda, ¿qué ocurre si hay varios accesos simultáneos a un determinado datos, uno para consultar y otro para actualizarlo? ¿Cómo se pueden gestionar la transacciones?



Un transacción es un conjunto de órdenes que se ejecutan formando un unidad de trabajo, en forma indivisible o atómica.

Para la gestión de transacciones en Hibernate, no se produce bloqueo de objetos en la memoria. La aplicación puede esperar el comportamiento definido por el nivel de aislamiento de sus transacciones de las bases de datos. Gracias a la **Session**, la cual también es un caché con alcance de transacción. Para realizar con éxito la gestión de transacciones, ésta se van a basar en el uso del objeto **Session**.



El objeto **Session** se obtiene a partir de un objeto **SessionFactory** ,invocando el método **openSession**. Un objeto **SessionFactory** representa una configuración particular de un conjunto de metadatos de mapping objeto/relaciona.

```
Session session = HibernateUtil.getSessionFactory().openSession();
Transaction tx = null;
try
{
    tx = session.beginTransaction();
    // Utilizar la Session para saveOrUpdate/get/delete/...tx.commit();
} catch (Exception e)
{
    if (tx != null)
    {
        tx.rollback();
        throw e;
    }
} finally {
    session.close();
} // Al finalizar la aplicación ...HibernateUtil.shutdown( );
```

Cuando se crea el objeto **Session** , se le asigna la conexión de la base de datos que va a utilizar. Una vez obtenido el objeto **Session** , se crea una nueva unidad de trabajo ( **Transaction**) utilizando el método **beginTransaction**. Dentro del contexto de la transacción creada, se pueden invocarlos métodos de gestión de persistencia proporcionados por el objeto **Session**, para recuperar, añadir, eliminar o modificar el estado de instancias de clases persistentes. También se pueden realizar consultas. Si las operaciones de persistencia no han producido ninguna excepción, se invoca el

método **commit** de la unidad de trabajo para confirmar los cambios realizados. En caso contrario, se realiza un **rollback** para deshacer los cambios producidos. Sobre un mismo objeto **Session** pueden crearse varias unidades de trabajo. Finalmente se cierra el objeto **Session** invocando su método **close**.

## Anexo.- Licencias de recursos.

### Licencias de recursos utilizados en la Unidad de Trabajo.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	Autoría: Verónica Cabrerizo. Licencia: GNU GPL. Procedencia: Captura de pantalla de la aplicación NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2.		Autoría: hibernate.org. Licencia: GNU LGPL. Procedencia: <a href="http://www.hibernate.org">http://www.hibernate.org</a>
	Autoría: ibatis.apache.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: <a href="http://ibatis.apache.org">http://ibatis.apache.org</a>		Autoría: Verónica Cabrerizo. Licencia: GNU GPL. Procedencia: Captura de pantalla aplicación NetBeans, propiedad Microsystems, bajo licencia GNU GPL
	Autoría: Verónica Cabrerizo. Licencia: GNU GPL. Procedencia: Captura de pantalla de la aplicación NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2.		Autoría: Verónica Cabrerizo. Licencia: GNU GPL. Procedencia: Captura de pantalla aplicación NetBeans, propiedad Microsystems, bajo licencia GNU GPL
	Autoría: Verónica Cabrerizo. Licencia: GNU GPL. Procedencia: Captura de pantalla de la aplicación NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2.		Autoría: Verónica Cabrerizo. Licencia: GNU GPL. Procedencia: Captura de pantalla aplicación NetBeans, propiedad Microsystems, bajo licencia GNU GPL
	Autoría: Verónica Cabrerizo. Licencia: GNU GPL. Procedencia: Captura de pantalla de la aplicación NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: <a href="http://netbeans.org/kb/docs/java/hibjava-se.html">http://netbeans.org/kb/docs/java/hibjava-se.html</a>
	Autoría: Verónica Cabrerizo. Licencia: GNU GPL. Procedencia: Captura de pantalla de la aplicación NetBeans, propiedad Sun Microsystems, bajo licencia GNU GPL v2.		