

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate



Developing Enterprise Java Applications with Hibernate

.es © noviembre 2016

1

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Hibernate: Introducción

.es © noviembre 2016

2



Developing Enterprise Java Application with Hibernate

Contenidos

- Introducción.
- ORM: Object Relational Mapping.
- Historia de los ORMs.

.es ©. Noviembre 2016

3



Developing Enterprise Java Application with Hibernate

Introducción

- ¿Qué aplicación de gestión no utiliza una BBDD?
- Un 20% del código de una aplicación son instrucciones relacionadas con la BBDD.
- El SQL rompe la POO, lo ideal sería que toda nuestra aplicación fuera POO, sin tener SQL.
- Necesidad de tener un framework que nos libere de las operaciones con la BBDD.

.es ©. Noviembre 2016

4

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

ORM

- "Object-Relational Mapping" (Mapeo Objeto Relacional) es una técnica de programación que resuelve el problema de "convertir" datos entre los sistemas de **tipos incompatibles** de un motor de **base de datos** relacional (RDBMS) y un lenguaje orientado a objetos, en nuestro caso **Java**.
- El **objetivo** es **crear una capa de persistencia** en la forma de una base de datos orientada a objetos (OODB) "virtual" que oculte al programador los detalles de persistencia de los datos en el RDBMS.
- Uno de los componentes ORM más utilizado (*sino el más...*) es **Hibernate**, surgido del ambiente **Java** y llevado al uso del framework **.NET** con la versión **Nhibernate**.

.es ©. Noviembre 2016

5

•es
Formación y consultoría

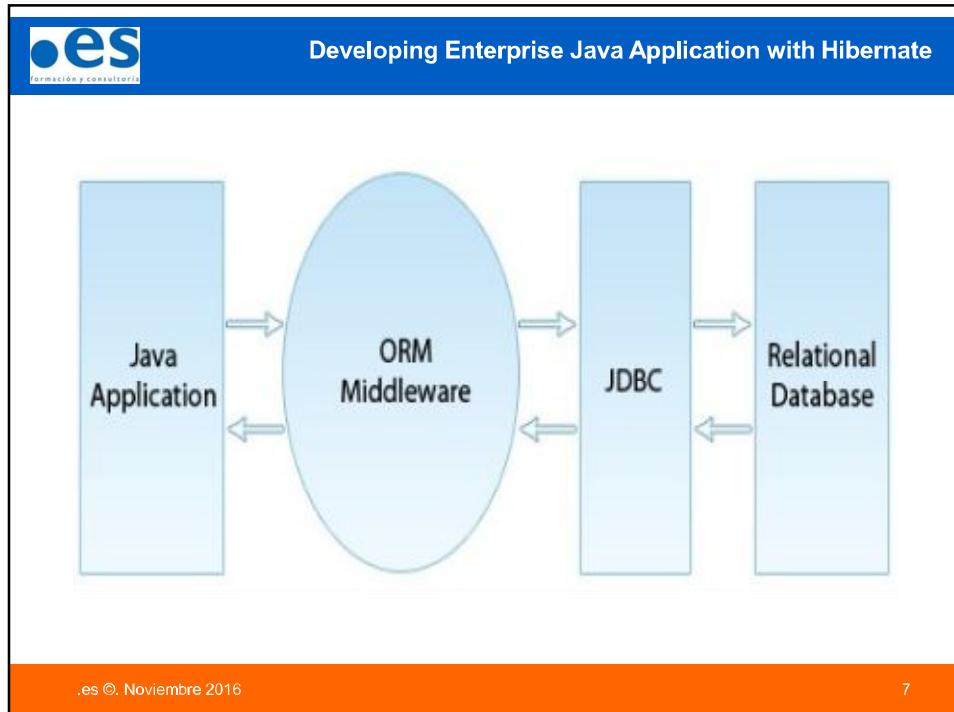
Developing Enterprise Java Application with Hibernate

Hibernate

- Hibernate es una herramienta desarrollada en Java.
- Se centra en la capa de **persistencia**.
- Es un **ORM**: Object Relational Mapping.
- Mapea tablas de la base de datos en objetos java.
- Desaparece el código SQL del programa.
- Nos centramos en la lógica de negocio.

.es ©. Noviembre 2016

6



Developing Enterprise Java Application with Hibernate

Hibernate

- El acceso a base de datos aparece prácticamente en todos los proyectos, por pequeños que sean.
- El acceso a base de datos es mecánico y repetitivo.
- Hibernate facilita esta tarea, nos podemos centrar en el modelo de Clases en vez de tablas de la Base de datos.

.es ©. Noviembre 2016

8

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Hibernate

- Trabajar con Hibernate implica:
 - No trabajar con filas de las tablas (ResultSet).
 - Trabajar con las clases diseñadas en su modelo del dominio.
 - Código OO limpio, permite trabajar con herencia y polimorfismo en nuestras clases de negocio.
 - Permite elegir la BD relacional con la que queremos: MySQL, Oracle, etc.
 - Genera automáticamente el código SQL usando un mapeo objeto-relacional.
 - Permite crear, modificar, recuperar y borrar objetos persistentes.

.es ©. Noviembre 2016

9

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Como definir el modelo

- El modelo se puede definir de dos formas:
- Mediante ficheros XML, de configuración.
 - Necesitamos un fichero XML que respalde cada clase Java.
- Mediante anotaciones, soportado a partir de Java 5.
 - Apuesta por la convención en vez de configuración.
- A la hora de mapear seleccionamos uno de los métodos, no se pueden mezclar.

.es ©. Noviembre 2016

10

 Developing Enterprise Java Application with Hibernate

Historia de los ORM

- En la comunidad Java se ha tenido claro que al programar es necesario usar un ORM y por ello se han creado gran cantidad de ORM.
- Entre estos **ORMs** encontramos:
 - **EJB** (*EnterPrise JavaBeans*),
 - **JDO** (*Java Data Object*),
 - **JPA** (*Java Persistence Api*)
 - **Hibernate**.

.es ©. Noviembre 2016 11

 Developing Enterprise Java Application with Hibernate

EJBs

- Aparece en 1999 el [JCP \(Java Community Process\)](#) ya empezó a estandarizar una tecnología de ORM llamada EJB 1.0 en el [JSR-19 \(Java Specification Request\)](#). Como siempre pasa con las versiones “1.0” y con ser una nueva tecnología , **fue un verdadero fracaso**.
- Los EJB eran una tecnología de ORM que **sólo podía usarse dentro de un Servidor de aplicaciones de Java EE**.

.es ©. Noviembre 2016 12

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

EJBs

- EJB 2.0 aparece en el año 2001 y con EJB 2.1 en el año 2003 con el [JSR-153](#) pero siguió siendo un fracaso.
- Un caso curioso fue el de **Rod Johnson**, que escribió un libro titulado [J2EE Development without EJB](#). Fue la primera vez que se vio un título de un libro que en vez de hablar de la tecnología que se debe usar, habla de tecnología que **NO** se debe usar.
- Realmente los EJB no eran tan inútiles pero no había tantos proyectos en los que fuera la tecnología correcta.
- Su mayor error fue intentar popularizarlos para que todo el mundo los usara. Aunque *quizás* dicho error fuera intencionado por parte de los vendedores de Servidores de Aplicaciones para vender sus productos.

.es ©. Noviembre 2016

13

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

JDO

- Por otro lado en el [JCP](#) también se estaba desarrollando otra especificación de ORM llamada JDO.
- JDO tenía varias ventanas respecto a los EJB:
 - Las clases a persistir no tenían que heredar de ninguna clase ni implementar ningún interfaz, lo que hacía transparente la persistencia a la base de datos. Es decir, que sean [POJOs](#)
 - No necesitaba de un Servidor de Aplicaciones para funcionar
 - Podía persistir a otro tipo de repositorios de datos que no fueran bases de datos relacionales

.es ©. Noviembre 2016

14

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Hibernate

- Con todo ello JDO era superior a los EJB, ¿qué pasó? Que JDO nunca tuvo el apoyo de las grandes empresas. Así que JDO es una tecnología que nadie usa.

Hibernate

- En **2001 Gavin King** creó Hibernate. Harto de los EJB, no se basó en ningún JSR ni ninguna especificación, simplemente creó un framework de ORM y la historia ha querido que Hibernate haya sido el ORM que mas éxito ha tenido.

.es ©. Noviembre 2016

15

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Historia de los ORM

- ¿Cuál ha sido el éxito de Hibernate?
 - No depende de estándares, lo que hace que pueda añadir funcionalidades más rápidamente.
 - Al igual que JDO no es necesario implementar interfaces o heredar de clases.
 - Realiza sólo una cosa y la hace bien: Sólo se puede persistir a bases de datos relacionales, a diferencia de JDO que permite otros tipo de repositorios.
- La principal desventaja es que al no haber una especificación detrás de él, si usas Hibernate ya no puedes cambiar de framework de persistencia tan fácilmente como podrías usando JDO o EJB.

.es ©. Noviembre 2016

16

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

JPA

- **Java Persistence API (*es el estándar*):**
 - Proporciona un estándar para **gestionar datos relacionales** en aplicaciones Java SE o Java EE, de forma que además se simplifique el desarrollo de la persistencia de datos.
 - Es un API de persistencia de **POJOs** (Plain Old Java Object). Es decir, objetos simples que no heredan ni implementan otras clases (como los EJBs).

.es ©. Noviembre 2016

17

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

JPA

- El **objetivo** que persigue en esta API es:
 - **No perder las ventajas de la orientación a objetos** al interactuar con una base de datos.
 - Sigue un patrón de **mapeo objeto-relacional**.
 - Permitir usar objetos regulares (**POJO**).
 - El mapeo objeto-relacional (**relaciona entidades Java y tablas de la Base de datos**) se puede realizar con **anotaciones** en las propias clases **evitando la configuración en XML**.

.es ©. Noviembre 2016

18

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

JPA

- Existen en el mercado varias implementaciones de JPA:
 - **Hibernate:**
<http://www.hibernate.org/>
 - **TopLink:**
<http://www.oracle.com/technetwork/middleware/toplink/overview/index.html>
 - **EclipseLink:**
<http://www.eclipse.org/eclipselink/>
 - **OpenJPA:**
<http://openjpa.apache.org/>
- La mas eficiente en cuanto a consumo de memoria y tiempo de acceso es **Hibernate**.

.es ©. Noviembre 2016

19

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Hibernate

- ¿Por qué un curso de Hibernate en vez de un curso de JPA?
Realmente este curso trata de Hibernate e Hibernate con JPA con lo que vamos a tener lo mejor de ambos.
- Para hacer la persistencia de nuestros objetos Java a la base de datos hay que indicar cómo se debe realizar dicha persistencia.
- Para ello hay dos métodos:
 - Ficheros XML
 - Anotaciones en el código.
- Las anotaciones que tiene Hibernate son actualmente las del estándar de JPA. De esa forma al ver las anotaciones de hibernate estaremos viendo las de JPA.

.es ©. Noviembre 2016

20

 **Developing Enterprise Java Application with Hibernate**

Hibernate

- Ofrece más ventajas del estándar de JPA.
- Los ficheros XML de mapeo son propietarios de Hibernate.
- Además no solo se encarga del mapeo de clases Java a tablas de la base de datos (y de regreso), sino que **también maneja las queries y recuperación de datos**, lo que puede reducir de forma significativa el tiempo de desarrollo que de otra forma gastaríamos manejando los datos de forma manual con **SQL y JDBC**

.es ©. Noviembre 2016 21

 **Developing Enterprise Java Application with Hibernate**

Hibernate

- De esta forma se encarga de alrededor del 95% de las tareas comunes relacionadas con la persistencia de datos, manejando todos los problemas relativos con la base de datos particular con la que estemos trabajando, de forma transparente para los desarrolladores.
- Entonces, **si cambiamos el manejador de base de datos no será necesario que modifiquemos todo el SQL** que ya teníamos para adaptarse al **SQL** que maneja la nueva base de datos. Solo será necesario **modificar una línea en un archivo de configuración de Hibernate**, y este se encargará del resto.

.es ©. Noviembre 2016 22

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Configuración

.es ©. Noviembre 2016

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Arquitectura

La arquitectura "completa" abstrae la aplicación de las APIs de JDBC/JTA y permite que Hibernate se encargue de los detalles

```
graph TD; Application[Application] --> SessionFactory[SessionFactory]; Application --> Transaction[Transaction]; SessionFactory --> Session[Session]; SessionFactory --> ConnectionProvider[ConnectionProvider]; Session --> PersistentObjects[Persistent Objects]; Session --> Transaction[Transaction]; Transaction --> JTA[JTA]; Transaction --> JDBC[JDBC]; Transaction --> JNDI[JNDI]; Database[Database]
```

The diagram illustrates the Java Persistence Architecture (JPA) stack. It consists of several layers: at the top is the Application layer, which contains Transient Objects and Persistent Objects; below it is the SessionFactory layer, which contains Session and Transaction components; further down are JNDI, JDBC, and JTA layers; at the bottom is the Database layer.

.es ©. Noviembre 2016

24

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Configuración

- Tenemos **dos** posibilidades de trabajar con Hibernate.
 - Descargar la distribución de:
<http://hibernate.org/orm/>
 - (*En la carpeta de software está descargada la distribución de Hibernate*).
 - O utilizar **Maven**.
- También necesitaremos el driver de la base de datos:
MySQL, Oracle, SQL-Server, etc.

.es ©. Noviembre 2016

25

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

IDEs

- **NetBeans**:
 - Ya viene con las librerías integradas y **NO** es necesario descargarlas.
 - Viene también con herramientas para generar las clases Java a partir del esquema de las Base de datos.
- **STS / Eclipse**:
 - Tendremos que utilizar los Jar descargados y que estén accesibles desde el path de nuestro proyecto (si no estamos utilizando maven).
 - Si queremos tener herramientas de generación automática de código tendremos que instalarlas.
 - Desde el menú de **Help → Eclipse MarketPlace (JBoss Tools)**.

.es ©. Noviembre 2016

26

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

NetBeans

En caso de utilizar la versión 8.x de Netbeans, ya viene integrada la librería de Hibernate dentro del IDE (**versión 4.3.x**)

Botón derecho sobre el proyecto

También viene el conector de MYSQL

.es ©. Noviembre 2016

27

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

STS / Eclipse

- Normalmente se puede crear una **carpeta** dentro del proyecto “lib” y ahí **copiamos** todos los jar que necesitemos.
 - En este caso si movemos el proyecto de máquina NO tendremos problemas con las librerías.
 - Si las librerías las tenemos fuera del proyecto, dentro del IDE podemos crear una “**librería de usuario**” para referenciarlas en cualquier proyecto que tengamos.
 - En este caso si movemos el proyecto a otra máquina si vamos a tener que crear la “librería de usuario” que referencia a todos los jar si no tendremos errores.
 - La opción **más cómoda** suele ser utilizar **Maven**, de forma automática se actualizará el repositorio local de nuestra máquina si es que no teníamos previamente las dependencias descargadas.

.es ©. Noviembre 2016

28

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Con Maven (STS / Eclipse / NetBeans)

- Independientemente del IDE que estemos utilizando en el fichero **POM.xml** del proyecto necesitaremos:
 - Dependencia de **Hibernate**:


```
<dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>4.3.11.Final</version>
          </dependency>
```
 - Y por ejemplo, con **MySQL**:


```
<dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.23</version>
          </dependency>
```

.es ©. Noviembre 2016 29

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Con Maven (STS / Eclipse / NetBeans)

- Hay veces que podemos tener problemas con alguna librería porque no se descarga correctamente o simplemente no la descarga.
- Utilizar:
 - Menú Proyect → Clean (en STS / Eclipse).
 - O Menú Run → Clean and Build Proyect (en NetBeans).
- En estos casos podemos optar por copiar directamente la librería en el repositorio local de nuestra máquina.
C:\Users\<><usuario>\.m2\repository
- Otras veces borrando el repositorio por completo, y lo descarga de nuevo.

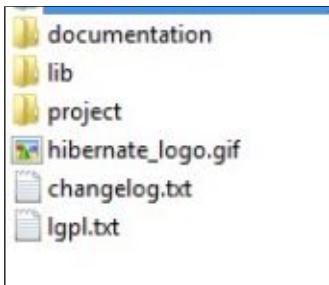
.es ©. Noviembre 2016 30

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Contenido de Hibernate

- Si hemos descargado y descomprimido la distribución de Hibernate:
- Veremos:



.es ©. Noviembre 2016

31

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Configuración de Hibernate

- En el fichero se encuentran 3 carpetas:
- **lib**: Contiene las librerías (*jars*) java con el código de Hibernate.
 - **lib\required**: Contiene los jars que siempre deben usarse en Hibernate. Es decir siempre debemos incluir estos jars en todos nuestros proyectos de Hibernate.
 - **lib\jpa**: Las librerías necesarias para usar JPA con Hibernate.
 - **lib\optional**: Contiene las librerías que añaden nuevas funcionalidades a hibernate, como poder usar el *Pool de conexiones C3PO* o el sistema de cache, etc.
 - **lib\envers**: Contiene librerías que permiten realizar auditorías sobre los datos que se persisten.

.es ©. Noviembre 2016

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Configuración de Hibernate

- **documentation:** Documentación sobre Hibernate.
 - documentation/devguide/en-US/html_single/index.html: Guía del desarrollador. “Hibernate Developer Guide”
 - documentation/quickstart/en-US/html_single/index.html: Guía de inicio. “Hibernate Getting Started Guide”
 - documentation/manual/en-US/html_single/index.html: Documentación de referencia sobre Hibernate. “Hibernate Reference Documentation”
 - documentation/javadocs/index.html: JavaDoc de las clases Java. “Hibernate JavaDoc”
- **project:** Contiene principalmente el código fuente y ficheros de configuración de las distintas bases de datos.

.es ©. Noviembre 2016

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Pool de Conexiones

- Hibernate por debajo utiliza conexiones JDBC, si utilizamos conexiones simples con la base de datos el coste en el rendimiento es caro.
- Una mejor solución es utilizar un pool de conexiones.
- Hibernate proporciona un pool de conexiones **C3P0** pero es muy rudimentario y no está pensado para un entorno en producción.
 - Se encuentra disponible en **lib/optional/c3p0**
- **Se debería de utilizar un pool 3 terceros registrado mediante JNDI en el servidor de aplicaciones.**

.es ©. Noviembre 2016

34

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Pool de Conexiones

- Si queremos utilizar este pool en un proyecto tenemos que incluir los jars correspondientes o las dependencias dentro del fichero pom.xml.
- En el fichero de configuración de hibernate tendríamos que indicar una propiedad:
`<property name="c3p0.timeout">10</property>`
- Maven:**
`<dependency>
 <groupId>org.hibernate</groupId>
 <artifactId>hibernate-c3p0</artifactId>
 <version>4.3.11.Final</version>
</dependency>`

.es ©. Noviembre 2016

35

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ficheros de Configuración Hibernate

- Dentro de un proyecto de Hibernate vamos a tener varios tipos de ficheros XML:
 - Ficheros de Mapeo: **NombreClase.hbm.xml**
 - Se utilizan para proporcionar toda la información necesaria para asociar objetos a las tablas de la Base de datos.
 - Normalmente se especifica un fichero por cada objeto que queremos mapear.
 - Cliente → Cliente.hbm.xml.
 - Fichero de Configuración: **hibernate.cfg.xml**
 - Información sobre la conexión (parámetros) y la ubicación de los ficheros de mapeo que vamos a utilizar.
 - Puede ser también un fichero .properties.
 - Fichero ingeniería inversa: **hibernate.reveng.xml**
 - Captura los nombres de las tablas a partir de un esquema de la BD.

.es ©. Noviembre 2016

36

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

NombreClase.hbm.xml

- Representan ficheros de mapeo de una tabla de la BD.
- Por regla general asocia, cada tabla de la BD a una clase Java y un fichero hbm.xml.
- En este fichero se especifica:
 - El nombre cualificado de la clase.
 - El nombre de la tabla de la BD.
 - El nombre del esquema de la BD.
 - Las columnas y el equivalente a las propiedades java.
 - Indicando cual es el campo clave.
 - Si es campo compuesto, genera una clase.
 - Las relaciones con otras entidades.
 - Indicando el tipo de relación, 1 a 1, 1 a muchos, etc.
 - Los campos relacionados.

.es ©. Noviembre 2016

37

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
<session-factory>
  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/empresa3?zeroDateTimeBehavior=convertToNull</property>
  <property name="hibernate.connection.username">root</property>
  <property name="hibernate.connection.password">antonio</property>
  <mapping resource="com/curso/hibernate/Clientes.hbm.xml"/>
  <mapping resource="com/curso/hibernate/Categorias.hbm.xml"/>
  <mapping resource="com/curso/hibernate/Empleados.hbm.xml"/>
  <mapping resource="com/curso/hibernate/Productos.hbm.xml"/>
  <mapping resource="com/curso/hibernate/Pedidos.hbm.xml"/>
  <mapping resource="com/curso/hibernate/Empresaservicios.hbm.xml"/>
  <mapping resource="com/curso/hibernate/Detallespedido.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

Este fichero almacena los parámetros de la conexión y la ubicación de los ficheros de mapeo. También tenemos que indicar el dialecto de la BD.

.es ©. Noviembre 2016

38

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

hibernate.reveng.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-reverse-engineering PUBLIC '-//Hibernate//DTD Reverse Engineering 3.0//EN' 'http://www.hibernate.org/dtd/hibernate-reverse-engineering_3_0.dtd'>
<hibernate-reverse-engineering>
  <schema-selection match-catalog="empresa3"/>
  <table-filter match-name="productos"/>
  <table-filter match-name="empresasenvios"/>
  <table-filter match-name="empleados"/>
  <table-filter match-name="detallespedido"/>
  <table-filter match-name="clientes"/>
  <table-filter match-name="categorias"/>
  <table-filter match-name="pedidos"/>
</hibernate-reverse-engineering>
```

Este fichero almacena todos los nombres de las tablas de la BD empresa3.

.es ©. Noviembre 2016

39

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Configuración Log4J

- No es obligatorio pero si aconsejable configurar Log4J para limitar los mensajes que emite hibernate por la consola.
- Con Log4J podremos enviar todas las trazas a un fichero de log o mostrarlas por la pantalla según nos interese.

.es ©. Noviembre 2016

40

Developing Enterprise Java Application with Hibernate

Configuración Log4J

- Dependencias **Maven**:

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
</dependency>

<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.6.4</version>
</dependency>
```

.es ©. Noviembre 2016 41

Developing Enterprise Java Application with Hibernate

Configuración Log4J

- También necesitamos crear un fichero:
log4j.properties
 - Que se ubicará en el mismo sitio que el fichero:
hibernate.cfg.xml
 - Dentro del fichero log4j mediante la configuración de propiedades indicaremos la configuración del log.

.es ©. Noviembre 2016 42



Developing Enterprise Java Application with Hibernate

Configuración Log4J

- **# Direct log messages to a log file**
- log4j.appenders.file=org.apache.log4j.RollingFileAppender
- #Path al fichero del log, utilizar \\
- log4j.appenders.file.File=C:\\mis logs\\app.log
- log4j.appenders.file.MaxFileSize=1MB
- log4j.appenders.file.MaxBackupIndex=1
- log4j.appenders.file.append=true
- log4j.appenders.file.layout=org.apache.log4j.PatternLayout
- log4j.appenders.file.layout.ConversionPattern=%d{ABSOLUTE} %5p
%c{1}:%L - %m%n

.es ©. Noviembre 2016

43



Developing Enterprise Java Application with Hibernate

Configuración Log4J

- **# Direct log messages to stdout**
- log4j.appenders.stdout=org.apache.log4j.ConsoleAppender
- log4j.appenders.stdout.Target=System.out
- log4j.appenders.stdout.layout=org.apache.log4j.PatternLayout
- log4j.appenders.stdout.layout.ConversionPattern=%d{ABSOLU
TE} %5p %c{1}:%L - %m%n

.es ©. Noviembre 2016

44



Developing Enterprise Java Application with Hibernate

Configuración Log4J

- **# Root logger option**
- #log4j.rootLogger=INFO, stdout, file
- # stdout → Escribe en Consola
- # file → Escribe en Fichero
- log4j.rootLogger=INFO, file
- # Log everything. Good for troubleshooting
- log4j.logger.org.hibernate=INFO
- #log4j.logger.org.hibernate = WARN
- #log4j.logger.org.hibernate = ERROR
- # Log all JDBC parameters
- log4j.logger.org.hibernate.type=ALL

.es ©. Noviembre 2016

45



Developing Enterprise Java Application with Hibernate

Ciclo de Vida

.es ©. Noviembre 2016



Developing Enterprise Java Application with Hibernate

Introducción

- Una de las mayores ventajas de Hibernate es que **trabaja** con objetos normales (**POJO**) sin tener que implementar algún interface especial o heredar de alguna clase de Hibernate.
- Es una forma de trabajar muy limpia.

.es ©. Noviembre 2016

47



Developing Enterprise Java Application with Hibernate

Estados de un objeto

- Hibernate contempla **3 posibles estados** de un objeto.
 - Transient (transitorio):
 - Persistent (Persistente):
 - Detached (Separado):

.es ©. Noviembre 2016

48



Developing Enterprise Java Application with Hibernate

Transient

- El objeto se acaba de instanciar con new y no está asociado a Session de Hibernate.
- No tiene una representación persistente en la base de datos.
- Y tampoco tiene asociado un valor identificador.
 - Por ejemplo, si la clave primaria fuera AI (autoincrement o identity), no tendría asociado un valor.
- Para convertirlo en persistente utilizaremos una Session de Hibernate.

.es ©. Noviembre 2016

49



Developing Enterprise Java Application with Hibernate

Persistent

- Una instancia persistente tiene una representación en la base de datos y un identificador asociado.
- Puede haber sido cargado o guardado.
- Se encuentra en el ámbito de una Session.

.es ©. Noviembre 2016

50



Developing Enterprise Java Application with Hibernate

Detached

- Una instancia separada es un objeto que se ha hecho persistente pero su Session ha sido cerrada.
- La referencia al objeto es todavía válida y se podría modificar su estado.
- Se puede re-unir a una Session haciéndola persistente de nuevo.

.es ©. Noviembre 2016

51



Developing Enterprise Java Application with Hibernate

Entidades & Clases

- Las entidades representan objetos Java que permiten almacenarlos en la Base de datos.
- Puede ocurrir que queramos tener un objeto Java de cara a los usuarios y que se reparta en dos tablas en la Base de datos.
- Como puede ser un departamento y sus N-Empleados, accesibles desde el mismo objeto.

.es ©. Noviembre 2016

52



Developing Enterprise Java Application with Hibernate

Entidades & Clases

- Normalmente una entidad quedará representada por una clase Java.
- Cada entidad estará representada por un nombre y este será el mismo que el de la clase Java.
- Hibernate da la opción de hacer cambios mediante los mapeos o anotaciones.

.es ©. Noviembre 2016

53



Developing Enterprise Java Application with Hibernate

Identificadores

- Un identificador o columna identificador, mapea el concepto de clave primaria en las bases de datos relacionales.
- Una clave primaria puede ser (una o mas columnas) se utilizan para especificar un dato particular dentro de un conjunto de datos.

.es ©. Noviembre 2016

54

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Identificadores

- Dos tipos:
 - Natural:
 - Algún dato que tiene sentido en la aplicación.
 - DNI, código empleado, número de la Seguridad Social.
 - Artificial:
 - Valores arbitrarios que la aplicación puede generar de forma secuencial como son los campos identidad.
- Comparativa:
 - Artificial menos ocupación de memoria, no se modifican ...

.es ©. Noviembre 2016

55

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Identificadores

- El tipo de clave dará lugar a **distintos tipos de estrategias** para la generación de la clave.
- Hay que tener en cuenta las capacidades de la base de datos (con la que estemos trabajando) para poder generar la clave.
- Por ejemplo, no todas las bases de datos soportan el **tipo secuencia**.

.es ©. Noviembre 2016

56

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Estrategias

- **Identity:**
 - Cuando se genera con un campo identidad.
- **Sequence:**
 - Secuencia, dependerá de la BD. Si lo soporta o no. Oracle, PostgreSQL, ...
- **Table:**
 - Este mecanismo utiliza una tabla para almacenar bloques de identificadores artificiales. En este caso habrá que utilizar una anotación especial (GenerationType.TABLE).
- **Auto:**
 - Normalmente se asocia con Identity depende de la BD en cuestión.
- **None:**
 - En si NO es una estrategia, cuando se intentara persistir un objeto se lanzaría una exception.

.es ©. Noviembre 2016

57

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

- Se indicarán con anotaciones o mapeos sobre las propiedades de la clase:

```

@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
Long id;

```

.es ©. Noviembre 2016

58

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Entidades & Asociaciones

- Las entidades pueden contener referencias a otras entidades ya sean embebidas como propiedades o indirectamente como una colección (*Arrays, Sets, Lists ...*)
- Estas asociaciones se representan por medio de la **clave foránea**, estas claves son identificadores que pueden participar en varias tablas.
 - Por eso que no interesa que las claves NO ocupen mucho (*identificadores artificiales*).

.es ©. Noviembre 2016

59

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Entidades & Asociaciones

- Las asociaciones pueden ser de dos tipos:
 - Unidireccionales:
 - Cuando una entidad contiene una referencia a otra entidad.
 - Bidireccionales:
 - Si la asociación es mutua.
- Un **error en Hibernate** es considerar **todas** las asociaciones **bidireccionales** cuando a lo mejor no tienen sentido a nivel del modelo.
 - Para ello disponemos de un lenguaje HQL donde podemos hacer esas asociaciones sin alterar el modelo.
 - Podemos tener problemas si se dan **dependencias circulares**.

.es ©. Noviembre 2016

60



Developing Enterprise Java Application with Hibernate

Tipos de Asociaciones

- **One to One:**
 - Cualquiera de los dos extremos puede ser el dueño (pero SOLO UNO, puede serlo). Si no lo hacemos así vamos a obtener dependencias circulares.
- **One to Many:**
 - La parte de Many debe ser el dueño.
- **Many to One:**
 - La parte de Many debe ser el dueño. En lo mismo pero desde la perspectiva opuesta.
- **Many to Many:**
 - Cada extremo de la asociación puede ser el dueño.

.es ©. Noviembre 2016

61



Developing Enterprise Java Application with Hibernate

Trabajo con objetos

.es ©. Noviembre 2016



Developing Enterprise Java Application with Hibernate

Contenidos

- Clases Principales.
- Operaciones de persistencia.
- Session
- SessionFactory

.es ©. Noviembre 2016

63



Developing Enterprise Java Application with Hibernate

Clases Principales

- Prácticamente todo el trabajo lo realizaremos a través de la interfaz **org.hibernate.Session** (podríamos comparar esta sesión con la conexión **java.sql.Connection** de **JDBC**).
- La sesión la obtenemos a través de una factoría: **org.hibernate.SessionFactory** (comparable con **java.sql.DriverManager**).
- **org.hibernate.Query** y **org.hibernate.Criteria** son dos clases que nos permiten, al igual que el **HQL** (Hibernate Query Language), personalizar las consultas a la base de datos (comparable con **java.sql.Statement**).
- **org.hibernate.Transaction**. Nos permite marcar los ámbitos de una transacción.

.es ©. Noviembre 2016

64

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Clases Principales

- La **SessionFactory** es costosa de obtener, se suele utilizar un patrón **Singleton** para obtener una única instancia a nivel de aplicación.
- La **Session** se suele abrir y cerrar por cada operación que realizamos sobre el modelo de objetos (save, delete, etc.).

.es ©. Noviembre 2016

65

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Clase HibernateUtil 4.3

```
public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            Configuration configuracion = new Configuration().configure();
            StandardServiceRegistryBuilder builder = new
                StandardServiceRegistryBuilder().applySettings(configuracion.getProperties());
            sessionFactory = configuracion.buildSessionFactory(builder.build());

        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

La **SessionFactory** se suele Obtener a partir de una clase Similar a esta. Utilizamos un **Singleton**

.es ©. Noviembre 2016

66

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Métodos

- Clase **SessionFactory**:
 - getCurrentSession()
 - openSession()
- Clase **Transaction**:
 - commit()
 - rollback()
- Clase **Session**:
 - beginTransaction()
 - **load() y get()**
 - **save(), update()**
 - **delete()**
 - **persist()**
 - **merge()**
 - refresh()
 - flush()
 - createCriteria()
 - createQuery(),
 - createSQLQuery()
 - getNamedQuery()
 - close()

.es ©. Noviembre 2016

67

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Esquema de las Transacciones

```
Transaction tx = null;
Session session = null;
try {
    session = HibernateUtil.getSessionFactory().openSession();
    tx = session.beginTransaction();
    // hacer algo con alguna entidad
    // ...
    tx.commit();
} catch (HibernateException e) {
    tx.rollback();
    throw e;
} finally {
    if (session != null) session.close();
}
```

La transacción la obtenemos
De la **session**.

.es ©. Noviembre 2016

68

Developing Enterprise Java Application with Hibernate

Utilización del código generado

- Nos olvidamos del SQL y creamos registros en la BD utilizando objetos:
- En el main(): (*Para crear un nuevo cliente en la BD*).
- Utilizar la clase generada: HibernateUtil.**

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
```

```
session.beginTransaction();
Cliente cliente = new Cliente();
cliente.setIdcliente("NUM_2");
cliente.setNombre("NOM_NUM2");
cliente.setPais("ES_es");
session.save(cliente);
session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
```

Pasos:

- Capturar la session.
- Abrir Transacción.
- Crear un objeto Cliente.
- Darle valores.
- Grabar.
- Hacer efectiva la transacción.
- Cerrar la session.

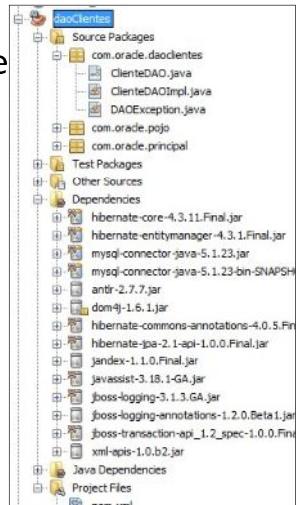
.es ©. Noviembre 2016

69

Developing Enterprise Java Application with Hibernate

Ejemplo paso a paso

- Programa que grabe en la tabla de clientes utilizando el API de Hibernate.
- Podemos utilizar Maven o crear una carpeta lib dentro de nuestro proyecto y copiar los jar de Hibernate además del driver de la BD que estemos utilizando.



.es ©. Noviembre 2016

70

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo paso a paso

- Necesitamos un **bean** que represente la clase **Cliente** (javabean → **constructor** sin parámetros y las propiedades **set /get**).
- Se corresponderá con la tabla de Clientes.
- No es obligatorio que los campos del objeto se llamen igual que los de la tabla.
- Asociado a este fichero llevamos un fichero XML en el que indicamos el mapeo del objeto en la tabla o también se puede anotar la clase.

.es ©. Noviembre 2016

71

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo paso a paso

- El fichero **Cliente.hbm.xml** se corresponde con la clase **Cliente.java**.
- Irán en la misma ubicación.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="entidades">
<class name="Cliente" table="clientes">
    <id name="idcliente" column="idcliente" ><generator class="assigned"/></id>
    <property name="nombre" />
    <property name="pais" />
</class>
</hibernate-mapping>
```

Indicar cual es la clave de la Tabla.
Con assigned la clave la asigna la app.

Si los campos de la tabla se llaman igual que los del objeto.
No hace falta indicarlo con el att. Column.

.es ©. Noviembre 2016

72

• El fichero de configuración de Hibernate: hibernate.cfg.xml

Las principales propiedades: los parámetros de la conexión con la BD:

```
<hibernate-configuration>
<session-factory>
<property name="connection.username">root</property>
<property name="connection.password">root</property>
<property name="connection.url">jdbc:mysql://localhost:3306/empresa3</property>
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>

<!-- Los ficheros de mapeo de la entidades -->
<mapping resource="entidades/Cliente.hbm.xml" />

</session-factory>
</hibernate-configuration>
```

.es ©. Noviembre 2016 73

• En el main():

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
Cliente cliente = new Cliente();
cliente.setIdcliente("NUM_2");
cliente.setNombre("NOM_NUM2");
cliente.setPais("ES_es");
session.save(cliente);
session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
```

Pasos:

- Capturar la sesión.
- Abrir Transacción.
- Crear un objeto Cliente.
- Darle valores.
- Grabar.
- Hacer efectiva la transacción.
- Cerrar la sesión.

Si el campo clave es identidad y se genera automáticamente, el método save lo devuelve.

Práctica 0

.es ©. Noviembre 2016 74



Developing Enterprise Java Application with Hibernate

Operaciones

- Una vez que el objeto está mapeado podemos realizar las operaciones: **CRUD**.
- Se aplican sobre el objeto **Session**:
 - Insertar → **save(MiObjeto)**;
 - Leer → **load / get(MiClase.class, valor_campo_clave)**;
 - Borrar → **delete(MiObjeto)**;
 - Actualizar → **update(MiObjeto)**;

.es ©. Noviembre 2016

75



Developing Enterprise Java Application with Hibernate

SessionFactory

- Proporciona instancias de **Session** que representan la base de datos.
- Se comparten por toda la aplicación pero sólo se pueden utilizar en una única transacción u unidad de trabajo.
- Esta clase se apoya en **Configuration** para poder crear la session.
- Hay **varias formas** de cargar la configuración.

.es ©. Noviembre 2016

76

Developing Enterprise Java Application with Hibernate

Configuration

- Con esta se carga el fichero **hibernate.cfg.xml**, por defecto busca este fichero en la carpeta **src** del proyecto. (LA MAS HABITUAL) o en la carpeta de **resources** si estamos utilizando **Maven**
- En el método **configure()** le podemos enviar la ruta de donde se encuentra el fichero **hibernate.cfg.xml**, si está en **src** o en **resources (Maven)** se deja vacío.

```
Configuration configuracion = new Configuration().configure();
StandardServiceRegistryBuilder builder = new
    StandardServiceRegistryBuilder().applySettings(configuracion.getProperties());
sessionFactory = configuracion.buildSessionFactory(builder.build());
```

.es ©. Noviembre 2016 77

Developing Enterprise Java Application with Hibernate

Configuration

- De **FORMA PROGRAMÁTICA** (indicando el fichero xml o el class)
- Si tenemos un fichero hbm se hace sobre el objeto configuracion: con **addFile**
`configuracion.addFile("com/manning/hq/ch03/Event.hbm.xml");`
- Si tenemos un class utilizaríamos el método **addAnnotatedClass**.
 - **OJO**, el **nombre cualificado** de la clase, indicando paquete y nombre de clase.
`configuracion.addAnnotatedClass(com.manning.hq.ch03.Event.class);`
- Utilizaremos también el método **setProperty()** para configurar propiedades de Hibernate.
- Los pasos antes y después para crear la configuración y obtener la **sessionFactory** son los mismos que antes.

.es ©. Noviembre 2016 78

Developing Enterprise Java Application with Hibernate

Ejemplo (carga configuración sin fichero hibernate.cfg.xml)

```

@Entity
@Table(name="clientes" ,catalog="empresa3")
public class Clientes implements java.io.Serializable {

    @Id
    @Column(name="idcliente", unique=true, nullable=false, length=10)
    private String idcliente;
    private String nombre;
    private String pais;
    private Boolean carnet;
    private Boolean musica;
    private Boolean deporte;
    private Boolean cine;
    private Boolean montanya;

    Métodos get / set / toString

```

Partimos de la siguiente clase
Que coincide con una Tabla de la Base de datos.

Si el atributo NO se anota con **@Column**
Se espera encontrar un campo de la Tabla Que se llama igual.

.es ©. Noviembre 2016

79

Developing Enterprise Java Application with Hibernate

Ejemplo (carga configuración sin fichero hibernate.cfg.xml)

```

Configuration cfg = new Configuration()
    .addAnnotatedClass(com.oracle.configuration.Clientes.class)
    .setProperty("hibernate.dialect", "org.hibernate.dialect.MySQLDialect")
    .setProperty("hibernate.connection.driver_class", "com.mysql.jdbc.Driver")
    .setProperty("hibernate.connection.username", "root")
    .setProperty("hibernate.connection.password", "antonio")
    .setProperty("hibernate.connection.url",
    "jdbc:mysql://localhost:3306/empresa3")
    .setProperty("hibernate.show_sql", "true")
    .setProperty("hibernate.format_sql", "true");

    StandardServiceRegistryBuilder builder = new
    StandardServiceRegistryBuilder().applySettings(cfg.getProperties());
    sessionFactory = cfg.buildSessionFactory(builder.build());

```

Abrir session, crear el objeto → Save (ojo meter Transacciones).

Práctica 12

.es ©. Noviembre 2016

80

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Propiedades de Configuración

- Dentro del fichero hibernate.cfg.xml se configura:
- Listado de ficheros de mapeo
- Dialecto de la BBDD. Engloba aquellas particularidades del SGBD.
- Cadena de conexión a la BBDD.
- Propiedades del pool de conexiones (C3PO, Apache DBCP, ...)
- *El pool de Hibernate NO es muy efectivo, no se suele usar para producción.*
- Propiedades adicionales:
 - **show_sql**: Muestras las sentencias SQL emitidas por el motor de persistencia
 - **format_sql**: Formatea las cadenas de las consultas para que sean legibles.

.es ©. Noviembre 2016

81

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Session – La unidad de Trabajo

- Session session = sessionFactory.openSession();
- Transaction tx = session.beginTransaction();
- En este punto el contexto de persistencia ha sido inicializado.
- La aplicación puede tener varios SessionFactory, cada uno conectado a una Base de Datos.
- La creación de SessionFactory es costosa, con lo que *se aconseja inicializar el SessionFactory en el arranque de la aplicación una única vez.*
- La obtención de Session es muy ligera, de hecho una *Session no obtiene la conexión JDBC hasta que no es necesario.*
- En la última línea se abre una transacción, y todas las operaciones que se ejecuten dentro de la unidad de trabajo se realizarán en la misma transacción.

.es ©. Noviembre 2016

82

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Persistiendo un objeto

- 1. Se instancia un objeto nuevo, se le asocian los valores a sus propiedades.
- 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 la BBDD, es necesario realizar el `commit` de la transacción. Dentro del objeto sesión se llama al método `flush()`. Es posible llamarlo explícitamente. *Si se ejecuta este método fuera de una transacción hay objetos en cascada estos no se grabarán a menos que vaciemos la session.*

En este momento, se obtiene la conexión JDBC a la BBDD para poder ejecutar la oportuna sentencia `INSERT`.

- 5. Finalmente, la sesión se cierra, con el objeto de liberar el contexto de persistencia.

.es ©. Noviembre 2016

83

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Recuperando un objeto

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

Item item = (Item) session.load(Item.class, new Long(1234));
// Item item = (Item) session.get(Item.class, new Long(1234));

tx.commit();
session.close();
```

- 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`.

Prueba Operaciones

.es ©. Noviembre 2016

84



OJO, con load()

- Al utilizar el método **load()** en vez del método **get()** podemos obtener el siguiente error:
- **Exception in thread "main"**
`org.hibernate.LazyInitializationException: could not initialize proxy - no Session.`
- Se soluciona cambiando el método **load** por **get**.



Modificando un objeto

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

Item item = (Item) session.get(Item.class, new Long(1234));
item.setDescription("This Playstation is as good as new!");

tx.commit();
session.close();
```

- No se llama directamente a **update**, hay que hacer primero un **get()**.
- Se modifican las propiedades.
- Y se actualiza.

 **Developing Enterprise Java Application with Hibernate**

Eliminar el objeto

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
session.delete(item);
tx.commit();
session.close();
```

Tener en cuenta que antes de borrar un objeto tendremos
Que cargarlo con get, normalmente irá en combinación.

.es ©. Noviembre 2016 87

 **Developing Enterprise Java Application with Hibernate**

Merge

- Merge se utiliza cuando tienes una instancia en estado de **Detached** (desconectada) y quieres que cambie a estado Persistente.
- Con los cambios actualizados en la base de datos.
- Merge se aplica sobre el objeto session.

.es ©. Noviembre 2016 88



Developing Enterprise Java Application with Hibernate

Merge

- Grabo un objeto dentro de su transacción.
- Cierro la Session y el objeto pasa a estado Detached.
- Modifico el objeto (una propiedad) y llamo a merge, Hibernate lanza un update para actualizar el objeto en la BD.

.es ©. Noviembre 2016

89



Developing Enterprise Java Application with Hibernate

Refresh

- Hibernate proporciona este método para refrescar objetos persistentes en la Base de datos.
- Recarga las propiedades de los objetos de una base de datos, sobrescribiendo las propiedades del objeto con los datos de la BD.

.es ©. Noviembre 2016

90

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Persist

- Hibernate persist es similar a guardar (con transacción) y agrega el objeto de entidad al contexto persistente, por lo que cualquier cambio adicional se rastrea.
- Si las propiedades del objeto se cambian antes de que se realice la transacción o se vacíe la sesión, también se guardará en la base de datos.

.es ©. Noviembre 2016 91

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Persist

```
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
//persist example - with transaction Session
session2 = sessionFactory.openSession();
Transaction tx2 = session2.beginTransaction();
Employee emp2 = HibernateSaveExample.getTestEmployee();
session2.persist(emp2);
System.out.println("Persist called");
emp2.setName("Kumar");
// will be updated in database too
System.out.println("Employee Name updated");
System.out.println("8. Employee persist called with transaction, id='"+emp2.getId()+"', address
    id='"+emp2.getAddress().getId(); tx2.commit();
System.out.println("*****");
// Close resources sessionFactory.close();
```

.es ©. Noviembre 2016 92

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

SaveOrUpdate

- Método de propósito general que bien **guarde** una instancia transitoria generando un identificador nuevo, o bien **actualice/reúna** las instancias separadas asociadas con su identificador actual.
- El método `saveOrUpdate()` implementa esta funcionalidad.
- Se utiliza cuando utilizamos instancias de una sesión en otra sesión.

.es ©. Noviembre 2016

93

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

flushing de la session

- Disponemos del método **flush()** con este método se fuerza a que los datos que están en la caché se vuelquen a la BD.
- Método:
 - `setFlushMode / getFlushMode()`
 - Los posibles modos son:
 - **ALWAYS**:
» Cada vez que se ejecuta una query se ejecuta flush → Bajo rendimiento y lentitud.
 - **AUTO**: (por defecto).
» Manejado por Hibernate, asegura los datos devueltos en una query están actualizados.
 - **COMMIT**:
» Hibernate hace flush al llamar a `commit()`.
 - **MANUAL**:
» Nuestra aplicación de forma manual tiene que llamar a `flush()`
- También dispone del método **clear()** para limpiar todos los objetos que se encuentran en la caché (se utiliza más en los procesos por lotes).

.es ©. Noviembre 2016

94

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Operaciones en Cascada

- Cuando se ejecutan operaciones sobre una entidad, **las operaciones no se ejecutarán sobre las entidades asociadas** a menos que se lo indiquemos a hibernate.
- Por ejemplo, tenemos una entidad producto que tiene una propiedad que representa una categoría.
- Si queremos grabar un producto, de una categoría que NO existe.

.es ©. Noviembre 2016

95

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Operaciones en Cascada

- El código sería:

```
session = HibernateUtil.getSessionFactory().openSession();
tx = session.beginTransaction();

cat = new Categorias("Informatica");
prod = new Productos();
prod.setCategorias(cat);
prod.setExistencias(10);
prod.setNombre("portatil");
prod.setPrecio(500.0F);

session.save(prod);
tx.commit();
```

Si no indicamos nada, Hibernate Lanzara una Exception porque El atributo categorias tiene valor null Está en un estado Transient no pertenece Al contexto de persistencia.

.es ©. Noviembre 2016

96

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Operaciones en Cascada

- Los tipos de cascada soportados por la arquitectura son:
 - PERSIST
 - MERGE
 - REFRESH
 - REMOVE
 - DETACH
 - ALL
 - Afecta a cualquier operación.

.es ©. Noviembre 2016 97

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Operaciones en Cascada

```
<class name="com.oracle.pojo.Productos" table="productos" catalog="empresa3" optimistic-lock="version">
    <id name="id" type="java.lang.Integer">
        <column name="id" />
        <generator class="identity" />
    </id>
    <many-to-one name="categorias" class="com.oracle.pojo.Categorias" fetch="select" cascade="all" />
    <property name="nombre" type="string">
        <column name="nombre" length="30" />
    </property>
    <property name="precio" type="java.lang.Float">
        <column name="precio" precision="12" scale="0" />
    </property>
    <property name="existencias" type="java.lang.Integer">
        <column name="existencias" />
    </property>
    <set name="detallespedidos" table="detallespedido" inverse="true" lazy="true" fetch="select">
        <key>
            <column name="idproducto" not-null="true" />
        </key>
        <one-to-many class="com.oracle.pojo.Detallespedido" />
    </set>
```

.es ©. Noviembre 2016 98

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Operaciones en Cascada

- Para el borrado ocurre algo similar:
- Si queremos eliminar una categoría y tiene productos asociados.
- Podríamos hacer:


```
session = HibernateUtil.getSessionFactory().openSession();
tx = session.beginTransaction();
cat = (Categorias) session.get(Categorias.class, new Integer(9));
session.delete(cat);
tx.commit();
System.out.println("Se ha borrado la categoria");
```
- En este caso nos dará un error de clave foránea.**

.es ©. Noviembre 2016 99

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Operaciones en Cascada

```
<class name="com.oracle.pojo.Categorias" table="categorias" catalog="empresa3" optimistic-lock="version">
  <id name="id" type="java.lang.Integer">
    <column name="id" />
    <generator class="identity" />
  </id>
  <property name="nombre" type="string">
    <column name="nombre" length="50" not-null="true" />
  </property>
  <set name="productoses" table="productos" inverse="true" lazy="true" fetch="select" cascade="delete">
    <key>
      <column name="idcategoria" not-null="true" />
    </key>
    <one-to-many class="com.oracle.pojo.Productos" />
  </set>
</class>
```

Al marcar en la asociación el atributo **cascade = "delete"** cuando se borre Una categoría realizará la operación en cascada con los productos asociados.

.es ©. Noviembre 2016 100

 Developing Enterprise Java Application with Hibernate

Carga tardía, proxies, Wrappers

- Cuando nos encontramos con BD con gran volumen de información y queremos recuperar información podemos tener problemas de memoria o bajo rendimiento.
- Por ejemplo, cargar una categoría con todos sus productos para mostrarlos en una página.
- Para manejar es tipo de problemas Hibernate proporciona una facilidad llamada “**lazy loading**”, por defecto se utiliza en los ficheros de mapeo pero no en las anotaciones.
- Una entidad asociada se cargará solo cuando sea directamente pedida.

.es ©. Noviembre 2016 101

 Developing Enterprise Java Application with Hibernate

Carga tardía, proxies, Wrappers

- La forma más simple que Hibernate puede forzar este comportamiento es con la implementación de un proxy.
- Hibernate intercepta la llamada por la sustitución de un proxy.
- Hibernate sólo puede acceder a la BD a través de la session.

.es ©. Noviembre 2016 102

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Carga tardía, proxies, Wrappers

- Si una entidad se desconecta de la session e intentamos acceder a ella a través de una asociación y la entidad todavía NO se ha cargado Hibernate lanza **LazyInitializationException**.
- Métodos static de **org.hibernate.Hibernate**:
 - Comprobaciones:
 - isInitialized(Object proxy)
 - isPropertyInitialized(Object proxy, String propertyName).
 - Forzar la inicialización:
 - initialize(Object proxy):

.es ©. Noviembre 2016

103

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Listados

- String sql = "from Trabajador";
- Query query = session.createQuery(sql);
- List trabajadores = query.list();
- Se apoya en el lenguaje HQL, lenguaje SQL sobre objetos.*(Lo veremos más en profundidad)*.

Práctica 2

.es ©. Noviembre 2016

104

The slide has a blue header bar with the logo '.es Formación y consultoría' and the title 'Developing Enterprise Java Application with Hibernate'. The main content area is white and contains the section title 'Mapeos' in bold black font. At the bottom of the slide is an orange footer bar with the text '.es ©. Noviembre 2016'.

The slide has a blue header bar with the logo '.es Formación y consultoría' and the title 'Developing Enterprise Java Application with Hibernate'. The main content area is white and contains the section title 'Introducción' in bold black font. Below it, a bulleted list describes the mapping process:

- El mapeo de objeto lo podemos hacer de dos formas.
 - Con archivos XML → MiClase.hbm.xml
 - Con Anotaciones → En la propia clase.
- Después en el fichero de configuración de Hibernate hay que indicar las entidades mapeadas:
 - Ejemplo **XML**:
 - <mapping resource="com/oracle/pojo/Productos.hbm.xml"/>
 - Ejemplo **Anotaciones**:
 - <mapping class="com.oracle.pojo.Empleados"/>

At the bottom of the slide is an orange footer bar with the text '.es ©. Noviembre 2016' and the page number '106'.

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Mapeo O/R

- Los mapeos objeto/relacional usualmente se definen en un documento XML. El documento de mapeo está diseñado para que se pueda leer y editar a mano.
- El lenguaje de mapeo está centrado en Java, lo que significa que los mapeos se construyen alrededor de declaraciones de clases persistentes y no alrededor de declaraciones de tablas.
- Existe herramientas de generación automática como NetBeans o Jboss Tools que se instalan en STS / Eclipse.

.es ©. Noviembre 2016 107

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Mapeo O/R

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="eg">

  <class name="Cat"
        table="cats"
        discriminator-value="C">

    <id name="id">
      <generator class="native"/>
    </id>

    <discriminator column="subclass"
                  type="character"/>

    <property name="weight"/>
    <property name="birthdate"
              type="date"
              not-null="true"
              update="false"/>
    <property name="color"
              type="eg.types.ColorUserType"
              not-null="true"
              update="false"/>
    <property name="sex"
              not-null="true"
              update="false"/>
    <property name="litterid"
              column="litterid"
              update="false"/>
  </class>
</hibernate-mapping>
```

.es ©. Noviembre 2016 108

Developing Enterprise Java Application with Hibernate

Mapeo O/R

- **Mapeo de Hibernate**
- Este elemento tiene varios atributos opcionales.
 - Los atributos **schema** y **catalog** especifican que las tablas a las que se refiere en este mapeo pertenecen al esquema y/o catálogo mencionado(s). De especificarse, los nombres de tablas serán calificados por el nombre del esquema y del catálogo dados. De omitirse, los nombres de las tablas no serán calificados.
 - El atributo **default-cascade** especifica qué estilo de cascada se debe asumir para las propiedades y colecciones que no especifican un atributo cascade.
 - El atributo **auto-import** nos permite utilizar nombres de clase sin calificar en el lenguaje de consulta.

.es ©. Noviembre 2016 109

Developing Enterprise Java Application with Hibernate

Mapeo O/R

```
<hibernate-mapping
    schema="schemaName"
    catalog="catalogName"
    default-cascade="cascade_style"
    default-access="field|property|ClassName"
    default-lazy="true|false"
    auto-import="true|false"
    package="package.name"
/>
```

1	schema (opcional): El nombre de un esquema de la base de datos.
2	catalog (opcional): El nombre de un catálogo de la base de datos.
3	default-cascade (opcional - por defecto es none): Un estilo de cascada por defecto.
4	default-access (opcional - por defecto es property): La estrategia que Hibernate debe utilizar para acceder a todas las propiedades. Puede ser una implementación personalizada de PropertyAccessor.
5	default-lazy (opcional - por defecto es true): El valor por defecto para los atributos lazy no especificados de mapeos de clase y de colección.
6	auto-import (opcional - por defecto es true): Especifica si podemos utilizar nombres de clases no calificados de clases en este mapeo en el lenguaje de consulta.
7	package (opcional): Especifica un prefijo de paquete que se debe utilizar para los nombres de clase no calificados en el documento de mapeo.

.es ©. Noviembre 2016 110

Developing Enterprise Java Application with Hibernate

Mapeo O/R

- Clase**
 - Puede declarar una clase persistente utilizando el elemento class. Por ejemplo:

```
<class
    name="ClassName"
    table="tableName"
    discriminator-value="discriminator_value"
    mutable="true|false"
    schema="owner"
    catalog="catalog"
    proxy="ProxyInterface"
    dynamic-update="true|false"
    dynamic-insert="true|false"
    select-before-update="true|false"
    polymorphism="implicit|explicit"
    where="arbitrary sql where condition"
    persister="PersisterClass"
    batch-size="N"
    optimistic-lock="none|version|dirty|all"
    lazy="true|false"
    entity-name="EntityName"
    check="arbitrary sql check condition"
    rowid="rowid"
    subselect="SQL expression"
    abstract="true|false"
    node="element-name"
/>
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

.es ©. Noviembre 2016

111

Developing Enterprise Java Application with Hibernate

Mapeo O/R

1	name (opcional): El nombre completamente calificado de la clase Java persistente (o interfaz). Si se omite este atributo, se asume que el mapeo es para una entidad que no es POJO.
2	table (opcional - por defecto es el nombre de la clase no calificado): El nombre de su tabla en la base de datos.
3	discriminator-value (opcional - predeterminado al nombre de la clase): Un valor que distingue subclases individuales, usado para el comportamiento polimórfico. Los valores aceptables incluyen null y not null.
4	mutable (opcional, por defecto es true): Especifica que las instancias de la clase (no) son mutables.
5	schema (opcional): Sobrescribe el nombre del esquema especificado por el elemento raíz <hibernate-mapping>.
6	catalog (opcional): Sobrescribe el nombre del catálogo especificado por el elemento raíz <hibernate-mapping>.
7	proxy (opcional): Especifica una interfaz a utilizar para los proxies de inicialización perezosa. Puede especificar el nombre mismo de la clase.
8	dynamic-update (opcional, por defecto es false): Especifica que el SQL UPDATE debe ser generado en tiempo de ejecución y puede contener sólamente aquellas columnas cuyos valores hayan cambiado.
9	dynamic-insert (opcional, por defecto es false): Especifica que el SQL INSERT debe ser generado en tiempo de ejecución y debe contener sólamente aquellas columnas cuyos valores no son nulos.
10	select-before-update (opcional, por defecto es false): Especifica que Hibernate <i>never</i> debe realizar un UPDATE SQL a menos de que se tenga certeza de que realmente se haya modificado un objeto. Sólo cuando un objeto transitorio ha sido asociado con una sesión nueva utilizando update()), Hibernate realizará una SQL SELECT extra para determinar si realmente se necesita un UPDATE.

.es ©. Noviembre 2016

112

Developing Enterprise Java Application with Hibernate

Mapeo O/R

(11) polymorphism (opcional, por defecto es implicit): Determina si se utiliza polimorfismo de consulta implícito o explícito.

(12) where (opcional) especifica una condición SQL WHERE arbitraria para utilizarla en la recuperación de objetos de esta clase.

(13) persister (opcional): Especifica un ClassPersister personalizado.

(14) batch-size (opcional, por defecto es 1) especifica un "tamaño de lote" para buscar instancias de esta clase por identificador.

(15) optimistic-lock (opcional, por defecto es version): Determina la estrategia optimista de bloqueo.

(16) lazy (opcional): La recuperación perezosa se puede deshabilitar por completo al establecer lazy="false".

(17) entity-name (optional - defaults to the class name): Hibernate3 allows a class to be mapped multiple times, potentially to different tables. It also allows entity mappings that are represented by Maps or XML at the Java level. In these cases, you should provide an explicit arbitrary name for the entity. See Sección 4.4, "Modelos dinámicos" and Capítulo 19, *Mapeo XML* for more information.

(18) check (opcional): Una expresión SQL utilizada para generar una restricción check multi-filas para la generación automática de esquemas.

(19) rowid (opcional): Hibernate puede utilizar los llamados ROWIDs en las bases de datos. Por ejemplo, en Oracle, Hibernate puede utilizar la columna extra rowid para actualizaciones rápidas si usted establece esta opción como rowid. Un ROWID es un detalle de implementación y representa la posición física de la tupla almacenada.

(20) subselect (opcional): Mapea una entidad inmutable y de sólo lectura a una subselección de base de datos. Es útil si quiere tener una vista en vez de una tabla base. Vea a continuación para obtener más información.

(21) abstract (opcional): Utilizado para marcar superclases abstractas en las jerarquías <union-subclass>.

Developing Enterprise Java Application with Hibernate

Mapeo O/R

- Es perfectamente aceptable que la clase persistente mencionada sea una interfaz. Puede declarar clases que implementan esa interfaz utilizando el elemento <subclass>.
- Se puede persistir cualquier clase interna *estática*.
- Debe especificar el nombre de la clase utilizando la forma estándar, por ejemplo, e.g.Foo\$Bar.

• Id

- Las clases mapeadas *tienen* que declarar la **columna de clave primaria** de la tabla de la base de datos.
- La mayoría de las clases también tendrán una propiedad de estilo Javabeans que tenga el identificador único de una instancia. El elemento <id> define el mapeo de esa propiedad a la columna de clave primaria.

.es ©. Noviembre 2016 115

```
<id
    name="propertyName"
    type="typename"
    column="column_name"
    unsaved-value="null|any|none|undefined|id_value"
    access="field|property|ClassName"
    node="element-name|@attribute-name|element/@attribute" .>

    <generator class="generatorClass"/>
</id>
```

1 name (opcional): El nombre de la propiedad del identificador. s
 2 type (opcional): un nombre que indica el tipo de Hibernate.
 3 column (opcional - por defecto es el nombre de la propiedad): El nombre de la columna de la clave principal.
 4 unsaved-value (opcional - por defecto es un valor "sensible"): Un valor de la propiedad identificadora que indica que una instancia está recién instanciada (sin guardar), distinguiéndola de las instancias separadas que fueron guardadas o cargadas en una sesión previa.
 5 access (opcional - por defecto es property): La estrategia que Hibernate debe utilizar para acceder al valor de la propiedad.

.es ©. Noviembre 2016 116

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Mapeo O/R

- El atributo unsaved-value casi nunca se necesita en Hibernate4.
- Hay una declaración <composite-id> opcional para permitir acceso a los datos heredados con claves compuestas.
- El elemento hijo opcional <generator> nombra una clase Java utilizada para generar identificadores únicos para instancias de la clase persistente. De requerirse algún parámetro para configurar o inicializar la instancia del generador, se pasa utilizando el elemento <param>.

.es ©. Noviembre 2016

117

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Mapeo O/R

```
<id name="id" type="long" column="cat_id">
    <generator class="org.hibernate.id.TableHiLoGenerator">
        <param name="table"
            >uid_table</param>
        <param name="column"
            >next_hi_value_column</param>
    </generator>
</id>
```

- Todos los generadores implementan la interfaz **org.hibernate.id.IdentifierGenerator**. Esta es una interfaz muy simple. Algunas aplicaciones pueden decidir brindar sus propias implementaciones especializadas. Sin embargo, Hibernate provee un rango de implementaciones ya incorporadas.

.es ©. Noviembre 2016

118

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Mapeo O/R

- Los nombres de atajo para los generadores incorporados son los siguientes:
 - **increment**: genera identificadores de tipo long, short o int que sólamente son únicos cuando ningún otro proceso está insertando datos en la misma tabla. *No lo utilice en un clúster.*
 - **identity**: soporta columnas de identidad en DB2, MySQL, MS SQL Server, Sybase y HypersonicSQL. El identificador devuelto es de tipo long, short o int.
 - **sequence**: usa una secuencia en DB2, PostgreSQL, Oracle, SAP DB, McKoi o un generador en Interbase. El identificador devuelto es de tipo long, short o int.
 - **hilo**: utiliza un algoritmo alto/bajo para generar eficientemente identificadores de tipo long, short o int, dada una tabla y columna como fuente de valores altos (por defecto hibernate_unique_key y next_hi respectivamente).

.es © Noviembre 2016 119

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Mapeo O/R

- **seqhilo**: utiliza un algoritmo alto/bajo para generar eficientemente identificadores de tipo long, short o int, dada una secuencia de base de datos.
- **uuid**: utiliza un algoritmo UUID de 128 bits para generar identificadores de tipo cadena, únicos dentro de una red (se utiliza la dirección IP). El UUID se codifica como una cadena hexadecimal de 32 dígitos de largo.
- **guid**: utiliza una cadena GUID generada por base de datos en MS SQL Server y MySQL.
- **native**: selecciona identity, sequence o hilo dependiendo de las capacidades de la base de datos subyacente.
- **assigned**: deja a la aplicación asignar un identificador al objeto antes de que se llame a save(). Esta es la estrategia por defecto si no se especifica un elemento <generator>.

.es © Noviembre 2016 120

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Mapeo O/R

- **select**: recupera una clave principal asignada por un disparador de base de datos seleccionando la fila por alguna clave única y recuperando el valor de la clave principal.
- **foreign**: utiliza el identificador de otro objeto asociado. Generalmente se usa en conjunto con una asociación de clave principal <one-to-one>.
- **sequence-identity**: una estrategia de generación de secuencias especializadas que utiliza una secuencia de base de datos para el valor real de la generación, pero combina esto junto con JDBC3 getGeneratedKeys para devolver el valor del identificador generado como parte de la ejecución de la declaración de inserción. Esta estrategia está soportada sólamente en los controladores 10g de Oracle destinados para JDK1.4. Los comentarios en estas declaraciones de inserción están desactivados debido a un error en los controladores de Oracle.

.es ©. Noviembre 2016

121

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo Clave Compuesta

```
<composite-id name="id"
  class="modelo.beans2.Detallespedidolid">

  <key-property name="idpedido" type="int">
    <column name="idpedido" />
  </key-property>

  <key-property name="idproducto" type="int">
    <column name="idproducto" />
  </key-property>

</composite-id>
```

.es ©. Noviembre 2016

122

Developing Enterprise Java Application with Hibernate

Mapeo O/R

- **Propiedad**
 - El elemento <property> declara una propiedad persistente estilo JavaBean de la clase.

```
<property name="propertyName"
column="column_name"
type="typename"
update="true|false"
insert="true|false"
formula="arbitrary SQL expression"
access="field|property|ClassName"
lazy="true|false"
unique="true|false"
not-null="true|false"
optimistic-lock="true|false"
generated="never|insert|always"
node="element-name|@attribute-
name|element/@attribute|."
index="index_name"
unique_key="unique_key_id"
length="L"
precision="P" scale="S" />
```

.es ©. Noviembre 2016 123

Developing Enterprise Java Application with Hibernate

Mapeo O/R

- ① name: el nombre de la propiedad, con la letra inicial en minúscula.
- ② column (opcional - por defecto es el nombre de la propiedad): El nombre de la columna de la tabla de base de datos mapeada. Esto se puede especificar también con los elemento(s) anidado(s) <column>.
- ③ type (opcional): un nombre que indica el tipo de Hibernate.
- ④ update, insert (opcional - por defecto es true): Especifica que las columnas mapeadas deben ser incluidas en las declaraciones SQL UPDATE y/o INSERT. Especificando ambas como false permite una propiedad "derivada", cuyo valor se inicia desde alguna otra propiedad que mapee a la misma columna (o columnas) o por un disparador u otra aplicación.
- ⑤ formula (opcional): una expresión SQL que define el valor para una propiedad computada. Las propiedades computadas no tienen una columna mapeada propia.
- ⑥ access (opcional - por defecto es property): La estrategia que Hibernate utiliza para acceder al valor de la propiedad.
- ⑦ lazy (opcional - por defecto es false): Especifica que se debe recuperar perezosamente esta propiedad cuando se acceda por primera vez la variable de instancia. Requiere instrumentación de código byte en tiempo de compilación.
- ⑧ unique (opcional): Activa la generación DDL de una restricción de unicidad para las columnas. Además, permite que ésta sea el objetivo de una property-ref.
- ⑨ not-null (opcional): Activa la generación DDL de una restricción de nulabilidad para las columnas.
- ⑩ optimistic-lock (opcional - por defecto es true): Especifica que las actualizaciones a esta propiedad requieren o no de la obtención de un bloqueo optimista. En otras palabras, determina si debe ocurrir un incremento de versión cuando la propiedad se encuentre desactualizada.

.es ©. Noviembre 2016 124

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones

- En Hibernate (y en general en las bases de datos) existen **4** tipos de relaciones:
 - **Uno a Uno**
 - **Uno a Muchos**
 - **Muchos a Uno**
 - **Muchos a Muchos**
- Si le ponemos direcciones a estas relaciones (**unidireccional**, o **bidireccional**) tendremos **7** tipos de relaciones:
 - **Uno a uno unidireccional**
 - **Uno a uno bidireccional**
 - **Uno a muchos unidireccional**
 - **Uno a muchos bidireccional**
 - **Muchos a uno unidireccional**
 - **Muchos a muchos unidireccional**
 - **Muchos a muchos bidireccional**
- *La relación muchos a uno bidireccional es igual que la relación uno a muchos bidireccional, así que la dejamos fuera.*

.es ©. Noviembre 2016

125

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Uno a Uno

- En las relaciones **uno a uno** un objeto entidad de una clase **A** está relacionado con **uno y solo un** objeto entidad de una clase **B**.
- Si la relación es **unidireccional** solo el objeto **A** está consciente de la relación (El objeto **A** tiene una referencia al objeto **B**) y el objeto **B** NO sabe nada de esta.
- Por ejemplo, cuando tenemos una relación **Persona -> Direccion** (es uno a uno porque una **Persona**, en teoría, solo puede tener una **Direccion**).
 - Donde la **Persona** es la entidad "A" y la **Direccion** es la entidad "B".
 - **Persona** conoce su **Direccion** pero la **Direccion** no conoce a la **Persona** a la que pertenece.

.es ©. Noviembre 2016

126

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

```
public class Direccion implements Serializable {
    private long id;
    private String calle;
    private String codigoPostal;
    Métodos Get / Set
}

public class Persona implements Serializable {
    private long id;
    private String nombre;
    private Direccion dirección;
}
```

- La clase Persona tiene acceso a la Dirección, esto implica que la clase **Persona es la dueña** (owner) de la relación.

.es ©. Noviembre 2016 127

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Uno a Uno

- Queremos que la entidad "Direccion" se guarde en la base de datos en el momento en el que se guarda la "**Persona**".
- De la misma forma, queremos que cuando la "**Persona**" sea eliminada de la base de datos también se elimine su correspondiente "**Direccion**". Para lograr esto usamos el atributo "**cascade**" del elemento "**<one-to-one>**".
- Las operaciones en cascada son operaciones que se realizan en los hijos al mismo momento que en los padres (o en las entidades relacionadas con la entidad en la que estamos realizando la operación)

.es ©. Noviembre 2016 128

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Uno a Uno - Bidireccional

- En las relaciones **bidireccionales** las dos entidades están conscientes de la relación.
- Por ejemplo, si tenemos una relación:
País <-> Presidente
- Es uno a uno porque un **País** solo debería tener un **Presidente** y un **Presidente** solo puede serlo de un **País**).
- En este caso ambos lados de la relación conocen el otro lado, o su "**inverso**".

.es ©. Noviembre 2016

129

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

```
public class Pais implements Serializable {
    private int id;
    private String nombre;
    private Presidente presidente;
    Get/Set
}

public class Presidente implements Serializable {
    private int id;
    private String nombre;
    private Pais pais;
    Get/Set
}
```

.es ©. Noviembre 2016

130

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Uno a Muchos

- En las relaciones **Uno a Muchos**, un objeto de la entidad "A" (el lado **uno**) está relacionado con muchos objetos de la entidad "B" (el lado **muchos**).
- En el caso de que la relación sea **unidireccional**, solo la entidad "A" tiene una referencia a los objetos de tipo "B" y esta relación está representada por una colección (un **List** o un **Set**) y la entidad "A" puede acceder a cada uno de los objetos de tipo "B" de esa colección.
- Un ejemplo de este tipo de relaciones puede ser una **Persona** puede tener muchos **Libros**. Donde la **Persona** es el lado **Uno** de la relación, y los **Libros** son el lado **Muchos**.
- Si la relación es **bidireccional**, adicionalmente, las entidades "B" tendrán una referencia a la entidad "A" con la que están relacionados. Un ejemplo de esto es un **Jefe** y sus **Empleados**. En donde el **Jefe** es el lado **Uno** y los **Empleados** el lado **Muchos**.

.es ©. Noviembre 2016

131

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

```
public class Libro {
    private long id;
    private String titulo;
    Get / Set
}

public class Persona {
    private long id;
    private String nombre;
    private List<Libro> libros = new
        ArrayList<Libro>();
}
```

.es ©. Noviembre 2016

132

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Uno a Muchos

- Para mapear una lista usamos el elemento "<list>". En este elemento indicamos cuál es el nombre del atributo, dentro de la clase **Persona**, que representa la relación. En este caso el atributo se llama "**libros**". También aquí indicamos cuáles operaciones queremos que se realicen en cascada. En este caso queremos que **todas** las operaciones de guardar, actualizar y eliminar que ocurran en el padre sean pasadas a la colección, o sea que cuando guardemos, actualicemos, o eliminemos una **Persona**, las operaciones pasen también a todos sus **Libros** relacionados, por lo que usamos el valor "**all**".
- En las relaciones **uno a muchos** existe dos estilos de cascada especiales llamados "**delete-orphan**" y "**all-delete-orphan**" (que solo existen si usamos archivos de mapeo) los cuales se encargan de que, en el caso de que se elimine el objeto padre (**Persona**), todos los objetos hijos (**Libros**) serán eliminados de la base de datos. Adicionalmente "**all-delete-orphan**" se encarga de que todas las otras operaciones que mencionamos antes (guardar, actualizar, y eliminar) también sean realizados en cascada, por lo que usaremos este valor:

.es ©. Noviembre 2016

133

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Uno a Muchos - Bidireccional

- Para crear la relación en los dos sentidos hay que modificar la entidad Libro añadiendo una referencia a la Persona.
- La clase Persona se mantiene tal cual.
- A nivel de mapeo tendremos que utilizar **many-to-one**

```
public class Libro {
    private long id;
    private String titulo;
    private Persona persona;
}
```

.es ©. Noviembre 2016

134

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Muchos a Uno

- Las relaciones muchos a uno en realidad son muy parecidas a las relaciones uno a muchos, con la excepción de que, en el caso de las relaciones unidireccionales, solo el lado muchos de la relación sabe de la existencia de esta.
- Es como si tuviéramos una clase **Televidente**, que tiene una relación **muchos a uno unidireccional** con una clase **CadenaTelevisiva**.
- Donde el **Televidente** (el lado **muchos** de la relación) tiene una **CadenaTelevisiva** favorita (el lado **uno** de la relación). Pero **CadenaTelevisiva** no tiene una referencia hacia cada uno de sus televidentes (desde los **Televidentes** podemos llegar a una **CadenaTelevisiva**, pero desde la **CadenaTelevisiva** no podemos llegar a los **Televidentes**).

.es ©. Noviembre 2016

135

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

```
public class CadenaTelevisiva {
    private long id;
    private String nombre;
}

public class Televidente {
    private long id;
    private String nombre;
    private CadenaTelevisiva cadenaFavorita;
}
```

Para representar la relación uno a muchos utilizamos el elemento **<many-to-one>**, En esta etiqueta colocamos, haciendo uso de se atributo name, **<many-to-one name="cadenaFavorita" />**

.es ©. Noviembre 2016

136

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Muchos a Muchos

- En ocasiones necesitamos relacionar muchas entidades de un tipo con muchas entidades de otro tipo. Este tipo de relaciones es muy común cuando trabajamos con bases de datos relacionales.
- Este tipo de relaciones es especial ya que, como mencione antes, tenemos que **muchos registros** de la entidad tipo A están relacionadas con **muchos registros** de la entidad tipo B.

.es ©. Noviembre 2016

137

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Muchos a Muchos

EntidadesA			
ID	Columna 1	Columna 2	LlaveForanea B
1	4
2	5
3	3
4	1

EntidadesB		
ID	Columna 1	Columna 2
1
2
3
4
5
6

.es ©. Noviembre 2016

138

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Muchos a Muchos

- Como en las relaciones **muchos a muchos** necesitamos relacionar **muchos** registros de la entidad **A**, con **muchos** registros de la entidad **B**, y es por esto que **NO** nos basta con una llave foránea en una de las tablas.
- Esto no puede ser logrado utilizando simplemente llaves foráneas en las tablas de las entidades **A** y **B**. En estos casos se utiliza una **tercera tabla** conocida como **tabla de join**, o **tabla de enlace**, o **tabla de unión**.

.es ©. Noviembre 2016

139

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Muchos a Muchos

The diagram illustrates a many-to-many relationship between two entities, EntidadesA and EntidadesB, using a junction table named UnionA-B. The EntidadesA table has columns ID, Columna 1, and Columna 2, with rows 1, 2, 3, and 4. The EntidadesB table has columns ID, Columna 1, and Columna 2, with rows 1 through 6. The UnionA-B table has columns LIF_A and LIF_B, with rows 1, 2, 3, and 4. Red arrows show the relationships: EntidadesA row 1 connects to UnionA-B rows 1 and 3; EntidadesA row 2 connects to UnionA-B rows 1 and 5; EntidadesA row 3 connects to UnionA-B rows 2 and 3; and EntidadesA row 4 connects to UnionA-B rows 3 and 4. EntidadesB rows 1 through 4 connect to UnionA-B rows 1 through 4 respectively, while EntidadesB rows 5 and 6 do not have any connections in the junction table.

EntidadesA		
ID	Columna 1	Columna 2
1
2
3
4

UnionA-B	
LIF_A	LIF_B
1	1
1	3
1	5
2	3
4	3

EntidadesB		
ID	Columna 1	Columna 2
1
2
3
4
5
6

.es ©. Noviembre 2016

140



Ejemplo

```
public class Materia {    public class Estudiante {
    private long id;          private long id;
    private String nombre;     private String nombre;
}
private List<Materia> materias =
new ArrayList<Materia>();
}
```



Relaciones Muchos a Muchos

- Para mapear una lista usamos el elemento "<list>". En este elemento indicamos cuál es el nombre del atributo, dentro de la clase **Estudiante**, que representa la relación. En este caso el atributo se llama "**materias**". También aquí indicamos cuáles operaciones queremos que se realicen en cascada. En este caso queremos que **todas** las operaciones de **guardar**, **actualizar** y **eliminar** que ocurran en el padre sean pasadas a la colección, o sea que cuando **guardemos**, **actualicemos**, o **eliminemos** un **Estudiante**, las operaciones pasen también a todas sus **Materias** relacionadas, por lo que usamos el valor "**all**".
- En las relaciones **muchos a muchos**, igual que en las relaciones **uno a muchos**, existen dos estilos de cascada especiales llamados "**delete-orphan**" y "**all-delete-orphan**" (que solo pueden usarse con archivos de mapeo) los cuales se encargan de que, en el caso de que se elimine el objeto padre ("**Estudiante**"), todos los objetos hijos ("**Materia**") serán eliminados de la base de datos. Adicionalmente "**all-delete-orphan**" se encarga de que todas las otras operaciones que mencionamos antes (**guardar**, **actualizar**, y **eliminar**) también sean realizados en cascada, por lo que usaremos este valor.

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Muchos a Muchos

- En las relaciones **muchos a muchos**, se usa una “**tabla de unión**” o “**tabla join**” para mantener los datos de qué objetos de la entidad A (**Estudiante**) están relacionados con qué objetos de la entidad B(**Materia**). En este caso debemos especificar cuál será el nombre de esta tabla de unión, usando el atributo “**table**” del elemento **<list>**.

```
<list name="materias" table="ESTUDIANTES_MATERIAS" cascade="all-delete-orphan">
    <key column="ID_ESTUDIANTE" />
    <list-index column="ORDEN" />
    <many-to-many
        class="hibernate.relaciones.muchos.muchos.modelo.Materia"
        column="ID_MATERIA" />
</list>
```

.es ©. Noviembre 2016 143

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Relaciones Muchos a Muchos Bidireccionales

```
public class Materia {
    private long id;
    private String nombre;
    private List<Estudiante> estudiantes
        = new ArrayList<Estudiante>();
}

public class Estudiante {
    private long id;
    private String nombre;
    private List<Materia> materias
        = new ArrayList<Materia>();
}
```

En el mapeo de Materia.hbm.xml

```
<list name="estudiantes" table="ESTUDIANTES_MATERIAS" inverse="true" >
    <key column="ID_MATERIA" />
    <list-index column="ORDEN" />
    <many-to-many class="hibernate.relaciones.muchos.muchos.modelo.Estudante"
        column="ID_ESTUDIANTE" />
</list>
```

.es ©. Noviembre 2016 144

Developing Enterprise Java Application with Hibernate

Mapeos con Anotaciones

Category	Annotations
Entity	@Entity
Database Schema Attributes	@Table @SecondaryTable @SecondaryTables @Column @JoinColumn @JoinColumns @PrimaryKeyJoinColumn @PrimaryKeyJoinColumns @JoinTable @UniqueConstraint
Identity	@Id @IdClass @EmbeddedId @GeneratedValue @SequenceGenerator @TableGenerator
Direct Mappings	@Basic @Enumerated @Temporal @Lob @Transient
Relationship Mappings	@OneToOne @ManyToOne @OneToMany @ManyToMany @MapKey @OrderBy

.es ©. Noviembre 2016 145

Developing Enterprise Java Application with Hibernate

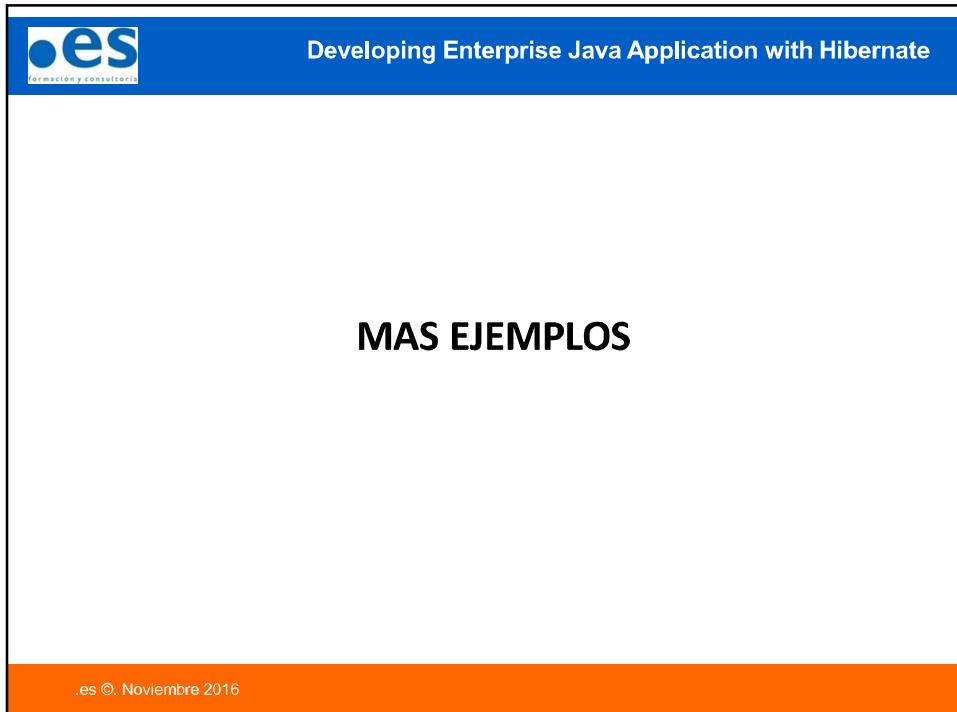
Ejemplo

```

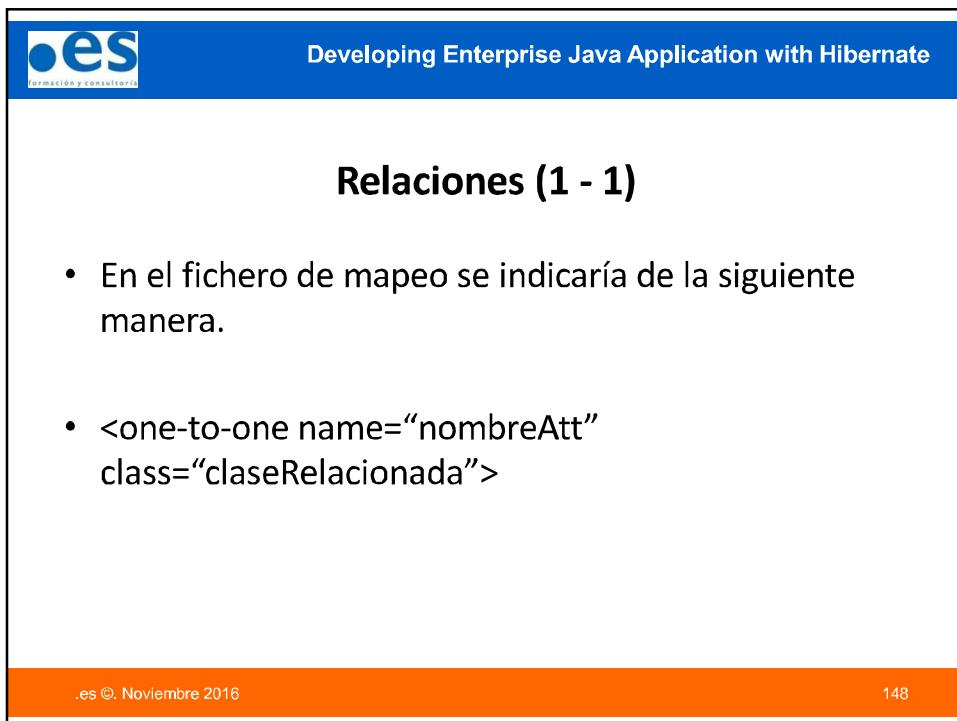
@Entity
@Table(name="Profesor")
public class Profesor implements Serializable {
    @Id
    @Column(name="Id")
    private int id;
    @Column(name="nombre")
    private String nombre;
    @Column(name="ape1")
    private String ape1;
    @Column(name="ape2")
    private String ape2;
}

```

.es ©. Noviembre 2016 146

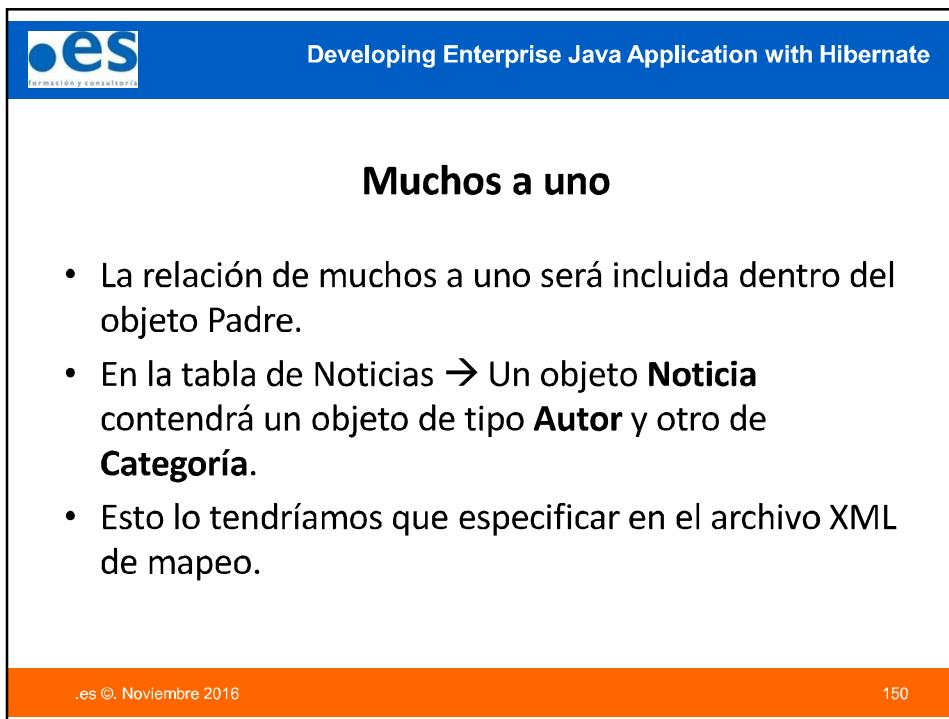
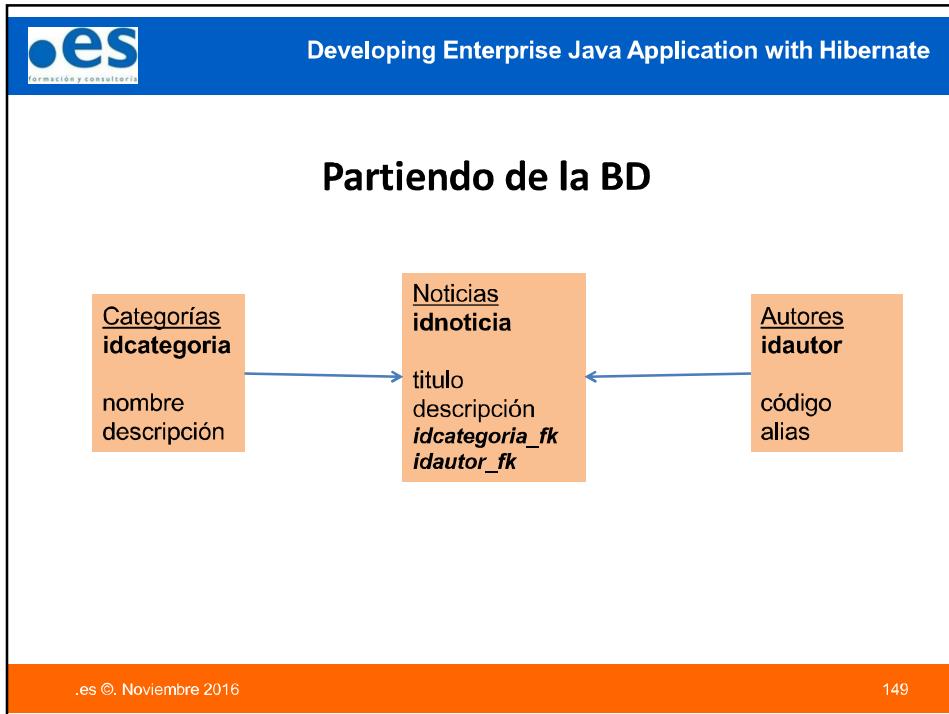


The slide features a blue header bar with the logo '.es formación y consultoría' on the left and the title 'Developing Enterprise Java Application with Hibernate' in white text on the right. The main content area is white and contains the section title 'MAS EJEMPLOS' in bold black capital letters. At the bottom of the slide is an orange footer bar with the text '.es ©. Noviembre 2016'.



The slide features a blue header bar with the logo '.es formación y consultoría' on the left and the title 'Developing Enterprise Java Application with Hibernate' in white text on the right. The main content area is white and contains the section title 'Relaciones (1 - 1)' in bold black capital letters. Below it is a bulleted list of two items. At the bottom of the slide is an orange footer bar with the text '.es ©. Noviembre 2016' on the left and the number '148' on the right.

- En el fichero de mapeo se indicaría de la siguiente manera.
- <one-to-one name="nombreAtt" class="claseRelacionada">



Developing Enterprise Java Application with Hibernate

Noticia.hbm.xml

```
<class name="com.anaya.java6.cp04.hibernate.modelo.Noticia" table="NOTICIAS">
    <id name="id" type="int">
        <column name="IDNOTICIA" />
        <generator class="identity" />
    </id>
    <property name="titulo" type="string">
        <column name="TITULO" length="40" not-null="true" />
    </property>
    <property name="descripcion" type="string">
        <column name="DESCRIPCION" length="260" not-null="true" />
    </property>
    <many-to-one name="categoria"
        class="com.anaya.java6.cp04.hibernate.modelo.Categoría" fetch="join">
        <column name="IDCATEGORIA_FK" not-null="true" />
    </many-to-one>
    <many-to-one name="autor"
        class="com.anaya.java6.cp04.hibernate.modelo.Autor" fetch="join">
        <column name="IDAUTOR_FK" not-null="true" />
    </many-to-one>
</class>
```

.es ©. Noviembre 2016 151

Developing Enterprise Java Application with Hibernate

Muchos a uno

- En este tipo de relación podríamos considerar el caso de las provincias y las comunidades.
- 1 Comunidad → N Provincias.
- *En este caso está relación se vería desde el punto de vista de la Provincia y dentro de esta tendríamos un objeto Comunidad.*

```
<class name="modelo.beans.Provincia" table="Provincias">
    <id name="id" type="int">
        <column name="idprovincia" />
        <generator class="identity" />
    </id>
    <property name="provincia" type="string">
        <column name="provincia" length="40" not-null="true" />
    </property>
    <many-to-one name="comunidad"
        class="modelo.beans.Comunidad" fetch="join">
        <column name="idcomunidad" not-null="true" />
    </many-to-one>
</class>
```

```
public class Provincia {
    private int id;
    private String provincia;
    private Comunidad comunidad;
    // Constructor / getters / setters
}
```

.es ©. Noviembre 2016 152

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Uno a Muchos

- Especifica la relación inversa de la anterior.
- Con relación a la Comunidades y las provincias estaríamos viendo la relación desde el punto de vista desde la Comunidad.
- Una Comunidad tendría un conjunto de Provincias.**
- Este tipo de relaciones pueden ser problemáticas desde el punto de vista de la memoria.
- ¿Qué ocurre si la tabla relacionada tiene miles de registros relacionados? → att. Lazy “Carga perezosa”*

.es ©. Noviembre 2016

153

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Autor.hbm.xml

```
<class name="com.anaya.java6.cp04.hibernate.modelo.Autor" table="AUTORES">
    <id name="id" type="int">
        <column name="IDAUTOR" />
        <generator class="identity" />
    </id>
    <property name="codigo" type="string">
        <column name="CODIGO" length="40" not-null="true" />
    </property>
    <property name="alias" type="string">
        <column name="ALIAS" length="40" not-null="true" />
    </property>
    <set name="noticias" inverse="true" lazy="true">
        <key>
            <column name="IDAUTOR_FK" precision="22" scale="0" />
        </key>
        <one-to-many class="com.anaya.java6.cp04.hibernate.modelo.Noticia" />
    </set>
</class>
```

```
public class Autor {
    private int id;
    private String codigo;
    private String alias;
    private Set<Noticia> noticias = new HashSet<Noticia>(0);
    public Autor() {}}
```

.es ©. Noviembre 2016

154

Partiendo de la BD

```

classDiagram
    class Categorias {
        idcategoria
        nombre
        descripción
    }
    class Noticias {
        idnoticia
        título
        descripción
        idcategoria_fk
        idautor_fk
    }
    class Autores {
        idautor
        código
        alias
    }
    Categorias <--> Noticias
    Noticias <--> Autores
  
```

```

public class Categoría {
    private int id;
    private String nombre;
    private String descripción;

    public Categoría(){}
    // Métodos set / get

    // Esta tb podría especificar un
    // Set de Noticias.
}

public class Noticia {
    private int id;
    private String título;
    private String descripción;
    private Categoría categoria;
    private Autor autor;

    public Noticia(){}
    // Métodos set / get
}

public class Autor{
    private int id;
    private String código;
    private String alias;
    private Set<Noticia> noticias = new HashSet<Noticia>(0);

    public Autor(){}
    // Métodos set / get
}
  
```

Práctica 4

.es ©. Noviembre 2016 155

Muchos a muchos (n-m)

- Esta relación nos permite tener relacionados una colección de objetos A dentro de un objeto B y viceversa.
- Esta relación representa la tabla relacional que surge de una relación de muchos a muchos. **Clientes (1) → (n) Ventas (m) ← (1) Libros**

```

<set name="A" table="A_B">
    <key>
        <column name="A_ID" />
    </key>
    <many-to-many class="B" column="B_ID"/>
</set>
<set name="B" table="A_B">
    <key>
        <column name="B_ID" />
    </key>
    <many-to-many class="A" column="A_ID"/>
</set>
  
```

.es ©. Noviembre 2016 156

Developing Enterprise Java Application with Hibernate

1 a muchos / muchos a 1

- Se pueden establecer las relaciones en ambos sentidos:

```

public class Categoria {
    private int id;
    private String nombre;
    private Set<Producto> productos = new HashSet<Producto>(0);

    <class name="Categoria" table="categorias">
        <id name="id" column="id" >
            <generator class="identity"/>
        </id>
        <property name="nombre" type="string">
            <column name="nombre" length="50" not-null="true" />
        </property>
        <set name="productos" inverse="true" lazy="true">
            <key>
                <column name="idcategoria" precision="22" scale="0" />
            </key>
            <one-to-many class="modelo.beans.Producto" />
        </set>
    </class>
}

```

El att. inverse=true
Indica que cuando se necesite buscar información se localice en la tabla de productos.

El att. lazy=true carga los datos de la colección cuando sea estrictamente necesario → cuando lo solicitemos. Si solo queremos el nombre de la categoría no se cargarían.

.es ©. Noviembre 2016

157

Developing Enterprise Java Application with Hibernate

1 a muchos / muchos a 1

```

public class Producto {
    private int id;
    private String nombre;
    private double precio;
    private int existencias;
    private Categoria categoria;

    <class name="Producto" table="productos">
        <id name="id" column="id" >
            <generator class="assigned" />
        </id>
        <property name="nombre" />
        <property name="precio" />
        <property name="existencias" />
        <many-to-one name="categoria" class="modelo.beans.Categoria" fetch="join">
            <column name="idcategoria" not-null="true" />
        </many-to-one>
    </class>
}

```

.es ©. Noviembre 2016

158

Developing Enterprise Java Application with Hibernate

Enlaces bidireccionales

```

classDiagram
    class EVENTS {
        *EVENT_ID
        EVENT_DATE
        TITLE
    }
    class PERSON_EVENT {
        *EVENT_ID
        *PERSON_ID
    }
    class PERSON {
        *PERSON_ID
        AGE
        FIRSTNAME
        LASTNAME
    }
    EVENTS <--> PERSON_EVENT
    PERSON_EVENT <--> PERSON
  
```

- En esta relación tenemos de **muchos eventos** ↔ **muchas personas**.
- En la clase **Event** tendríamos una colección de las personas que participan.



```

private Set participants = new HashSet();
public Set getParticipants() {
    return participants;
}
public void setParticipants(Set participants) {
    this.participants = participants;
}
  
```

`<set name="participants" table="PERSON_EVENT" inverse="true">
<key column="EVENT_ID"/>
<many-to-many column="PERSON_ID" class="events.Person"/>
</set>`

- El atributo **inverse=true** indica a Hibernate que debe tomar el otro lado, la clase **Person**, cuando necesite encontrar información sobre el enlace entre las dos.

.es ©. Noviembre 2016 **Práctica 5** 159

Developing Enterprise Java Application with Hibernate

Agregar elementos a las colecciones

```

classDiagram
    class PERSON {
        *PERSON_ID
        AGE
        FIRSTNAME
        LASTNAME
    }
    class PERSON_EMAIL_ADDR {
        *PERSON_ID
        *EMAIL_ADDR
    }
    PERSON <--> PERSON_EMAIL_ADDR
  
```

- En la relación **1 a n**:
 - **1 persona** tiene **n correos**.
 - La clase Person tendría un conjunto (Set) de emails
 - Para agregar un email a la Persona, mediante código:

```

private void addEmailToPerson(Long personId, String emailAddress) {
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
    Person aPerson = (Person) session.load(Person.class, personId);

    aPerson.getEmailAddresses().add(emailAddress);
    session.getTransaction().commit();
}
  
```

Práctica 6

.es ©. Noviembre 2016 160

Developing Enterprise Java Application with Hibernate

Operaciones bidireccionales

- En el ejemplo de las personas y los eventos, si queremos mantener las relaciones en ambos sentidos tendríamos que hacer (en la clase Person):

```

protected Set getEvents() {
    return events;
}

protected void setEvents(Set events) {
    this.events = events;
}

public void addToEvent(Event event) {
    this.getEvents().add(event);
    event.getParticipants().add(this);
}

public void removeFromEvent(Event event) {
    this.getEvents().remove(event);
    event.getParticipants().remove(this);
}

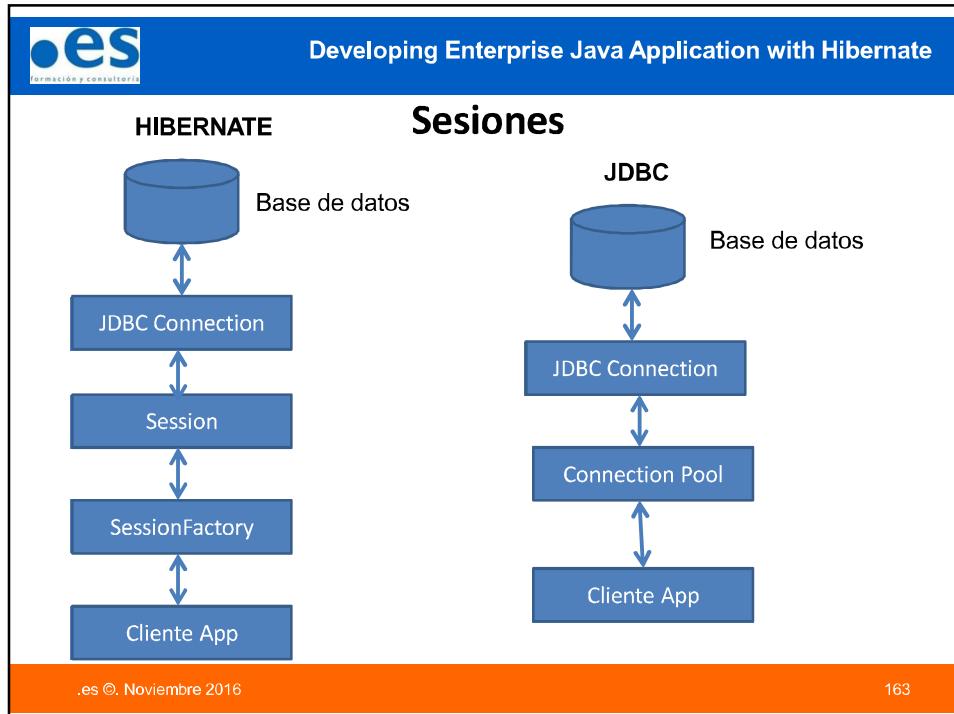
```

.es ©. Noviembre 2016 161

Developing Enterprise Java Application with Hibernate

Transacciones y Concurrencia

.es ©. Noviembre 2016



•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Métodos de Session

- **save()** :
 - Graba un objeto en la BD, no utilizar si el objeto ya existe.
- **saveOrUpdate()**:
 - Graba o Actualiza. Más costoso que save, ya que utiliza un select para comprobar si existe el objeto.
- **merge()**:
 - Fusiona los campos de un objeto no persistente en el objeto persistente apropiado (determinado por ID). Si no existe tal objeto en la base de datos, entonces uno se crea y se guarda.
- **persist()**:
 - Re-asocia un objeto con la session (por ejemplo, si estaba en estado detach), de modo que los cambios se harán persistentes.
- **get()**:
 - Recupera un objeto de la base de datos a partir del identificador.

.es ©. Noviembre 2016 165

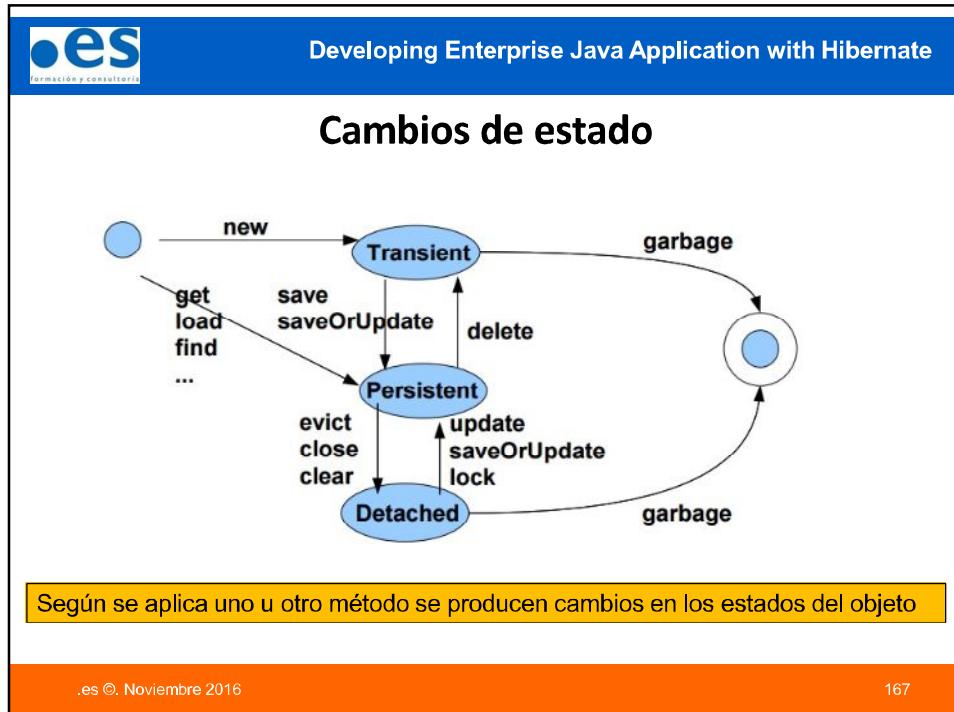
•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Métodos de Session

- **getEntityName()**:
 - Recupera el nombre de la entidad (por lo general, será el mismo que el nombre cualificado de la clase).
- **getIdentifier()**:
 - Devuelve la clave primaria del objeto.
- **load()**:
 - Carga un objeto de la BD, mejor utilizar get.
- **refresh()**:
 - Refresca el estado de un objeto asociado con la BD.
- **update()**:
 - Actualiza la BD con los cambios del objeto.
- **delete()**:
 - Borrar el objeto de la BD.

.es ©. Noviembre 2016 166



•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Métodos de Session

- **beginTransaction():**
 - Inicio de transacción.
- **getTransaction():**
 - Recuperar la transacción en curso.
- **lock():**
 - Devuelve el bloqueo de la BD para un objeto.
- **contains():**
 - Determina si un objeto está asociado a la BD.
- **clear():**
 - Borra la sesión de todas las instancias cargadas y cancela grabaciones, actualizaciones o eliminaciones que no se han completado.
- **evict():**
 - Desasocia un objeto de la session y los cambios no se harán persistentes.

.es ©. Noviembre 2016

169

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Métodos de Session

- **flush():**
 - Vuelca todos los cambios pendientes con la BD. Sincroniza la BD con la session.
- **isOpen():**
 - Determina si la session está o no abierta.
- **isDirty():**
 - Determina si la session está sincronizada con la BD.
- **getCacheMode():**
 - Recuperar el modo de cache empleada.
- **setCacheMode():**
 - Establecer el modo de cache
- **getCurrentLockMode():**
 - Determina el modo de bloqueo.
- **setFlushMode():**
 - Establece el modo de Flush.
- **setReadOnly():**
 - Establece el modo de lectura para un objeto.

.es ©. Noviembre 2016

170

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Métodos de Session

- **close():**
 - Cerrar la session. No hacer operaciones con la session después de cerrar. Libera otros recursos como la caché.
- **getSessionFactory():**
 - Recupera la SessionFactory que creo la Session actual.
- **connection():**
 - Recupera una referencia a la conexión subyacente.
- **disconnect():**
 - Desconectar la conexión subyacente.
- **reconnect():**
 - Reconectar la conexión subyacente con la BD
- **isConnected()**
 - Determina si está conectada con la conexión subyacente.

.es ©. Noviembre 2016

171

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Transacciones

- Las **transacciones** son operaciones de **todo o nada**.
- Nos permiten agrupar varias operaciones en una única unidad de trabajo que si va todo bien se completa o si no se rechazan los cambios.
- Las transacciones garantizan que los datos y los recursos nunca queden en estado de inconsistencia.
- Toda aplicación empresarial debería hacer uso de transacciones.

.es ©. Noviembre 2016

172

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Transacciones

- Las transacciones y los bloqueos están íntimamente relacionados.
- Las técnicas de bloqueo determinan el éxito de la transacción y el rendimiento.
- No es obligatorio utilizar transacciones pero si hay que utilizarlas para que las cosas funcionen correctamente.
- Si no se usan habría que llamar al método **flush** para que los cambios surtan efecto en la BD.

.es ©. Noviembre 2016

173

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Esquema de las Transacciones

```

graph LR
    IS((Initial State)) -- begin --> T[Transaction]
    T -- commit --> TC((Transaction completed))
    T -- roll back --> TF((Transaction failed))
  
```

.es ©. Noviembre 2016

174

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Transacciones

```
Session session = factory.openSession();
try {
    session.beginTransaction();
    // Operaciones con la sesión.

    session.getTransaction().commit();

} catch (HibernateException e){
    Transaction tx = session.getTransaction();
    if (tx.isActive()) tx.rollback();

} finally {
    if (session != null) session.close();
}
```

.es ©. Noviembre 2016 175

.es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Características A.C.I.D

- **Atómicas:**
 - Están formadas **por una o más actividades agrupadas** entre sí formando una única unidad de trabajo.
- **Coherentes / Consistencia:**
 - Una vez **finalizada** una transacción (sea con éxito o no), el sistema queda en un estado **coherente**.
- **Independientes / Aislamiento:**
 - Deben permitir que **varios usuarios puedan trabajar con los mismos datos** sin que el trabajo de cada usuario se vea afectado por el de otros.
- **Duraderas:**
 - Cuando termina una transacción, **los resultados se deben hacer permanentes**.

.es ©. Noviembre 2016 176

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Niveles de Aislamiento

- Definen que parte de una transacción va a verse afectada por las actividades de otras transacciones que tengan lugar a la vez.
- El trabajo simultáneo sobre los datos puede dar lugar a:
 - **Lectura de datos sucios:**
 - Una tx lee datos que han sido escritos por otra tx pero no se han aplicado. Si los cambios se cancelan los datos de la primera tx no serán válidos.
 - **Lectura no repetible:**
 - Una tx realiza la misma consulta 2 o más veces y cada vez los datos son diferentes, puede ocurrir porque otra tx está actualizando los datos entre las consultas.
 - **Lectura fantasma:**
 - Una tx lee varias filas y a continuación otra inserta filas, después de varias consultas la primera tx encuentra filas que no estaban ahí.

.es ©. Noviembre 2016

177

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Niveles de Aislamiento JDBC

Permitidos en Hibernate:

- 0 **None**
 - Cualquier cosa se permite
- 1 **Read Uncommitted**
 - Lectura de datos sucios, no repetible y fantasma.
- 2 **Read Committed**
 - Lectura no repetible y fantasma.
- 4 **Repeatable Read**
 - Lectura fantasma
- 8 **Serializable**
 - Evita todas las situaciones anteriores.

En casi todas las BBDD suele ser **read committed** o **repeatable read**

```
<property name="hibernate.connection.isolation">2</property>
```

.es ©. Noviembre 2016

178

Ejemplo

```

public static void createUser(String username){
    // Estableciendo un nivel de aislamiento.
    Session session = factory.openSession();
    int isolation = -1;

    try {
        isolation = session.connection().getTransactionIsolation();
        session.connection().setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
        session.beginTransaction();

        // Operaciones con la sesión.

        session.getTransaction().commit();

    } catch (SQLException e){
        rollback(session);
        throw new HibernateException(e);

    } catch (HibernateException e){
        rollback(session);
        throw e;

    } finally{
        reset(session, isolation);
        if (session != null) session.close();
    }
}

```

.es ©. Noviembre 2016

Si queremos utilizar el Nivel de aislamiento por Defecto no tenemos que Hacer cambios en la conexión

179

Ajuste nivel Aislamiento

- Si se pone nivel muy restrictivo se ralentiza mucho el acceso.
- Si se pone muy bajo aparecerán datos inconsistentes difíciles de depurar.

.es ©. Noviembre 2016

180

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Escoger el nivel de Aislamiento

- Depende de necesidades, pero:
 - **Read uncommitted nunca** (solo expertos)
- **Serializable no es frecuente**
 - ¿Realmente me van a afectar los phantom reads? → pocas TX son así.
- La duda mayor está entre **read committed** y **repeatable read**
 - Read committed no protege del second lost update y sí puede ser importante.
 - Repetible read sí protege frente a second lost update
 - Pero no lo tienen todas las BDD (Oracle no lo tiene, PostgreSQL tampoco, ...)

.es ©. Noviembre 2016

181

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Bloqueos

- Por defecto Hibernate trabaja con **bloqueo optimista**.
- Esto quiere decir que no se hacen bloqueos, pensando que es raro que dos usuarios trabajen exactamente sobre el mismo registro en el mismo instante de tiempo.
- El bloqueo optimista garantiza la escalabilidad de la aplicación.
- Hibernate proporciona bloqueo optimista comprobando **la versión del registro o el timestamp**.

.es ©. Noviembre 2016

182

Developing Enterprise Java Application with Hibernate

Bloqueos

- Para el bloqueo pesimista: siempre se usa el bloqueo de la base de datos subyacente.
- La clase **LockMode** define los distintos niveles de bloqueo disponibles en Hibernate:
 - LockMode.WRITE** se adquiere al insertar o actualizar una fila.
 - LockMode.UPGRADE** se puede conseguir explícitamente usando `SELECT ... FOR UPDATE`, en las bases de datos que soportan esta sintaxis.
 - LockMode.UPGRADE_NOWAIT** se puede conseguir en Oracle usando `SELECT ... FOR UPDATE NOWAIT`.
 - LockMode.READ** se adquiere al realizar lecturas sobre un nivel de aislamiento: Repeatable Read o Serializable. También se puede adquirir por petición expresa del usuario.
 - LockMode.NONE** todos los objetos se cambian a este modo de bloqueo una vez termina la transacción.
- Para definir el **LockMode** se hará una llamada a:
 - `session.load();`
 - `session.lock();`
 - `query.setLockMode(string, unLockMode);`

Read Uncommitted
Bloqueo optimista

Serializable
Bloqueo pesimista

.es ©. Noviembre 2016 183

Developing Enterprise Java Application with Hibernate

Cache

- El **acceso a la BD es costoso**, y por esta razón Hibernate trabaja con una **cache** para facilitar la recuperación de esos objetos.
- Hibernate funciona con una **caché de primer nivel** y otra **caché de segundo nivel**.

```

graph TD
    BD([Base de datos]) <-->|Hibernate| L1[L1 Cache]
    L1 <-->|Hibernate| Session[Session]
    Session -->|Hibernate| Cliente[Cliente]
    L2[L2 Cache] -- "Opcional" --> Session
  
```

.es ©. Noviembre 2016 184

 **Developing Enterprise Java Application with Hibernate**

Cache de Primer Nivel

- Se mantiene automáticamente por Hibernate.
- Dentro de una transacción se mantienen en memoria los objetos que fueron cargados.
- Si mas adelante se necesitan van a ser retornados desde el cache, ahorrando accesos sobre la base de datos.
- No requiere configuración.

.es ©. Noviembre 2016 185

 **Developing Enterprise Java Application with Hibernate**

Cache de Primer Nivel

- Se puede **deshabilitar** en que caso de NO queramos utilizar caché.
- Tipo especial de Session: **StatelessSession**.
 - SessionFactory.openStatelessSession()
 - Se utiliza en los **procesos batch** para hacer cargas masivas.
 - Esto evita excepciones del tipo **OutOfMemoryError**.

.es ©. Noviembre 2016 186

 Developing Enterprise Java Application with Hibernate

Cache de Segundo Nivel

- La diferencia fundamental es que éste tipo de cache **es válido para todas las transacciones** y puede persistir en memoria durante todo el tiempo en que el **aplicativo esté online**.
- Funciona como un **cache global**.
- Es **externo a Hibernate** pero **transparente** para el usuario.

.es ©. Noviembre 2016 187

 Developing Enterprise Java Application with Hibernate

Cache de Segundo Nivel

- Proveedores de cache de 2º Nivel:
 - EhCache
 - <http://www.ehcache.org/>
 - Infinispan
 - Sucesor de Jboss Cache
 - OsCache
 - SwarmCache

.es ©. Noviembre 2016 188

 **Developing Enterprise Java Application with Hibernate**

Configuración EhCache

- Pasos:
 - Seleccionar un proveedor de cache.
 - Dependencias Maven
 - Configurar hibernate para utilizar la cache de 2º nivel.
 - Modificar la propiedades de hibernate.cfg.xml
 - Indicar cual son las entidades a cachear.

 **Developing Enterprise Java Application with Hibernate**

Dependencias Maven

- Segundo elijamos uno u otro proveedor:
 - <dependency>
 - <groupId>org.hibernate</groupId>
 - <artifactId>hibernate-ehcache</artifactId>
 - <version>4.3.11.Final</version>
 - </dependency>

 Developing Enterprise Java Application with Hibernate

Propiedades Hibernate

```
<property name="hibernate.cache.use_second_level_cache">true
</property>

<property name="hibernate.cache.use_query_cache">true
</property>

<property name="hibernate.cache.region.factory_class">
    org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory
</property>
```

.es ©. Noviembre 2016 191

 Developing Enterprise Java Application with Hibernate

Anotar las clases

```
@Entity
@Cacheable
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY)
public class MiEntidad implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    Integer id;
    @Column(unique=true)
    String name;
}
```

.es ©. Noviembre 2016 192

The slide features a blue header bar with the logo '.es formación y consultoría' on the left and the title 'Developing Enterprise Java Application with Hibernate' in white text on the right. The main content area is white and contains the section title 'Interceptores & Eventos' in bold black font. At the bottom of the slide is an orange footer bar with the text '.es ©. Noviembre 2016'.

The slide features a blue header bar with the logo '.es formación y consultoría' on the left and the title 'Developing Enterprise Java Application with Hibernate' in white text on the right. The main content area is white and contains a section titled 'Introducción' in bold black font. Below it is a bulleted list of points explaining the need for interceptors and events. At the bottom of the slide is an orange footer bar with the text '.es ©. Noviembre 2016' on the left and the page number '194' on the right.

- Algunas veces podemos tener situaciones que demanden la realización de algunas operaciones antes o después de nuestra lógica funcional (precondiciones y postcondiciones).
- Si queremos intervenir antes o después de qué alguna de nuestras operaciones de persistencia (alta, baja, actualización, lectura, etc.) sea realizada.
 - Por ejemplo, actualización de existencias en un almacén.
- También algunas veces es necesario recibir alguna notificación de algún suceso que esté ocurriendo en nuestro motor de persistencia, como el estar recuperando o eliminando algún objeto. Esto puede ser útil para propósitos de auditorías, o para obtener estadísticas sobre las operaciones de persistencia en nuestras aplicaciones.

Hibernate proporciona dos mecanismos para lograr estos dos objetivos: **interceptores y eventos**.

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Interceptores

- Los interceptores nos proporcionan llamadas, a **nivel sesión** o a **nivel aplicación**, permitiendo a la aplicación inspeccionar y/o manipular propiedades de un objeto persistente antes de ser guardado, actualizado, eliminado, o cargado dentro de nuestro contexto persistente.
- Pueden ser utilizados para monitorear los eventos ocurridos o para sobrescribir la funcionalidad de un módulo.
- El ejemplo clásico es la auditoría del sistema, para realizar un log de eventos que indiquen los cambios que realizan sobre nuestras entidades.

.es ©. Noviembre 2016

195

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Interceptores

- Hibernate dispone de la interface:
 - org.hibernate.Interceptor**
 - Proporciona 18 métodos que tienen que ver con las operaciones que realizamos sobre una entidad.
 - En concreto son 13 métodos los más importantes o utilizados.
 - Cuando queramos crear un interceptor haremos:
 - public class MiInterceptor implements Interceptor {}**

.es ©. Noviembre 2016

196

Developing Enterprise Java Application with Hibernate

Métodos de un Interceptor

- **afterTransactionBegin** – Este método es llamado inmediatamente después de que una transacción es iniciada.
- **afterTransactionCompletion** – Llamado al terminar una transacción.
- **beforeTransactionCompletion** – Llamado antes de que se realice un **commit** de la transacción (solo de **commit**, no de **rollback**).
- **findDirty** – Llamado en el momento de llamar a "flush()".
- **onCollectionRecreate** – Llamado antes de que una colección se creada o recreada.
- **onCollectionRemove** – Llamado antes de que una colección sea eliminada.
- **onCollectionUpdate** – Llamado antes de que una colección sea actualizada.
- **onDelete** – Llamado antes de que una instancia sea eliminada
- **onLoad** – Llamado antes de que una entidad sea inicializada (o sea antes de establecer los valores de sus atributos).
- **onPrepareStatement** – Llamado cuando se está preparando la cadena con el SQL generado.
- **onSave** – Llamado antes de que una entidad sea almacenada.
- **postFlush** – Llamado después del **flush** que sincroniza los datos con la base de datos.
- **preFlush** – Llamado antes de un **flush**.

.es ©. Noviembre 2016 197

Developing Enterprise Java Application with Hibernate

Interceptores

- Otra forma que proporciona Hibernate de crear un Interceptor es heredando de la clase:
org.hibernate.EmptyInterceptor
- Ejemplo:

```
public class InterceptorAuditoria extends
EmptyInterceptor { }
```

.es ©. Noviembre 2016 198

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

```
public class InterceptorAuditoria extends EmptyInterceptor {

    @Override public boolean onSave(Object entity,
        Serializable id, Object[] state, String[] propertyNames,
        Type[] types) { }

    @Override public void onDelete(Object entity, Serializable
        id, Object[] state, String[] propertyNames, Type[] types) {
    }

}
```

.es ©. Noviembre 2016

199

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

onSave

- Es llamado **antes** de que un **objeto sea guardado**.
- Devuelve un **boolean** para indicar si hemos modificado el objeto.
 - Si hemos modificado alguno de los valores del objeto.
 - Si no necesitamos esa información devolveremos false.
 - Dentro del método haremos una comprobación si el objeto object es una instancia de la clase (la que vamos a interceptar).

.es ©. Noviembre 2016

200

Ejemplo

```
public boolean onSave(Object entity, Serializable id, Object[] state,
    String[] propertyNames, Type[] types) {
    if(entity instanceof Usuario) {
        Usuario usuario = (Usuario)entity; System.out.println("Se ha
            almacenado al Usuario " + usuario.getNombre() + ",\"" +
            usuario.getUsername() + "\"");
    }
    return false;
}
```

onDelete

- Cada vez que un **Usuario** sea eliminado se debe mostrar un mensaje en consola".
- Utilizaremos el método "**onDelete**":
`void onDelete(Object entity, Serializable id, Object[]
 state, String[] propertyNames, Type[] types) throws
 CallbackException`



onDelete

- **onDelete** se llama antes de que la entidad sea eliminada.
- No devuelve nada, no tiene sentido modificar los atributos del objeto.
- Parámetros:
 - **entity**, que es la entidad que se va a eliminar,
 - **id**, que es el identificador de dicha entidad en la base de datos.
- Dentro del método haremos un casting a la clase para ello preguntaremos antes con el operador **instanceof**.



Ejemplo

```
public void onDelete(Object entity, Serializable id, Object[]
    state, String[] propertyNames, Type[] types) {
    if(entity instanceof Usuario) {
        Usuario usuario = (Usuario)entity;
        System.out.println("Se eliminará al Usuario " +
            usuario.getNombre() + ", id=" + id);
    }
}
```

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Instalación del Interceptor

- Cuando abrimos la session tenemos que indicar a Hibernate los interceptores:
- Aplicamos un patrón Builder:


```
— session = sessionFactory.withOptions().interceptor(new MiInterceptor()).openSession();
```

PRACTICA 13

.es ©. Noviembre 2016 205

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Eventos

- Hibernate proporciona una arquitectura de **eventos**.
- Cada método de **Session** está relacionado con un evento, de esta forma podemos recibir notificaciones cuando se producen llamadas de estos métodos.
- Para cada evento tenemos una interface que tendremos que implementar para recibir notificaciones.

.es ©. Noviembre 2016 206

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Eventos

- Todos los eventos se encuentran en el paquete:
org.hibernate.event.spi
- Tenemos eventos “**pre**” que se lanzan antes de ejecutar el método y eventos “**post**” que se lanzan después de lanzar el método.
- Por ejemplo,
 - LoadEventListener
 - MergeEventListener

.es ©. Noviembre 2016 207

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Eventos

- Ejemplo de implementación:
 - **Antes** de que un **Usuario** sea cargado debemos mostrar un mensaje en consola.
 - PostLoadEventListener
 - **Antes** de eliminar un **Usuario** debemos mostrar un mensaje en consola.
 - PreDeleteEventListener
 - **Después** de eliminar un **Usuario** debemos mostrar un mensaje en consola.
 - PostDeleteEventListener
 - **Antes** de actualizar un **Usuario** debemos mostrar un mensaje en consola.
 - PreUpdateEventListener

.es ©. Noviembre 2016 208

Developing Enterprise Java Application with Hibernate

Ejemplo

```
public class CargaUsuarioListener implements
    PostLoadEventListener {
    public void onPostLoad(PostLoadEvent postLoadEvent) {
        Object entidad = postLoadEvent.getEntity();
        if (entidad instanceof Usuario) {
            Usuario usuario = (Usuario) entidad;
            System.out.println("Se ha cargado el usuario " +
                usuario.getUsername() + ", id=" + +
                postLoadEvent.getId());
        }
    }
}
```

.es ©. Noviembre 2016 209

Developing Enterprise Java Application with Hibernate

Ejemplo

- Una mejor solución es implementar un patrón Adapter.

```
public class HibernateEventAdapter implements
    PreInsertEventListener,
    PostInsertEventListener,
    PreUpdateEventListener,
    PostUpdateEventListener,
    PreDeleteEventListener,
    PostDeleteEventListener {}
```

- Una clase que implemente todos los interfaces que necesitemos.
- Dejando los métodos vacíos.
- Y otra clase que herede de esta sobrescribiendo los métodos que nos interesen.

```
public class ProductoEventAdapter extends HibernateEventAdapter {}
```

.es ©. Noviembre 2016 210

Developing Enterprise Java Application with Hibernate

Configuración

- Una vez tenemos los eventos programados hay que configurarlos.
- Esto se podía hacer dentro del fichero de configuración (hibernate.cfg.xml) en hibernate 3, en hibernate 4 cambia.
- De echo si se declaran los eventos dentro del fichero de configuración con la etiqueta **event se ignoran en hibernate 4**
- **Tenemos que crear una clase que implemente el interface Integrator y en el método integrate hacemos el registro.**

.es ©. Noviembre 2016 211

Developing Enterprise Java Application with Hibernate

Integrator

```
public class HibernateIntegrator implements Integrator {

    @Override public void integrate(Configuration configuration,
SessionFactoryImplementor sessionFactory, SessionFactoryServiceRegistry
serviceRegistry) {
    final EventListenerRegistry eventListenerRegistry =
    serviceRegistry.getService(EventListenerRegistry.class);

    // Crear una instancia de la clase que maneja los eventos:
    ProductoEventAdapter pea = new ProductoEventAdapter();

    // Registrarlos, EventType dispone de todos los tipos de eventos ...
    eventListenerRegistry.setListeners(EventType.PRE_INSERT, pea);
    eventListenerRegistry.setListeners(EventType.PRE_UPDATE, pea);
    eventListenerRegistry.setListeners(EventType.PRE_DELETE, pea);
    eventListenerRegistry.setListeners(EventType.POST_INSERT, pea);
    eventListenerRegistry.setListeners(EventType.POST_UPDATE, pea);
    eventListenerRegistry.setListeners(EventType.POST_INSERT, pea);
}

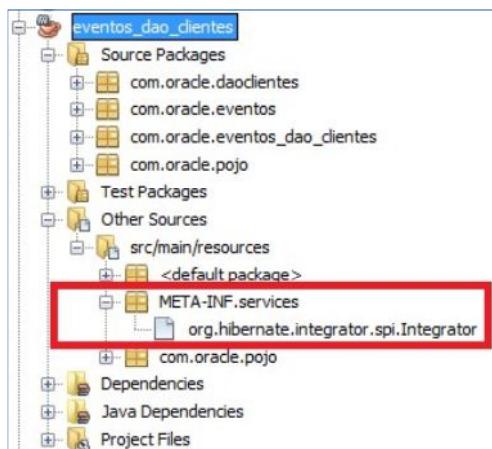
.es ©. Noviembre 2016 212
```

Integrator

- Por último, necesitamos un fichero de texto en la siguiente ubicación:
`META-INF/services/org.hibernate.integrator.spi.Integrator`
 - Se ubica en la carpeta de recursos (Maven).
- Dentro de este fichero indicamos el nombre cualificado de la clase **Integrator**:
 - Por ejemplo:
`com.oracle.eventos.HibernateIntegrator`

.es ©. Noviembre 2016 213

Integrator



PRACTICA 14

.es ©. Noviembre 2016 214



Developing Enterprise Java Application with Hibernate

Procesamiento por Lotes

.es ©. Noviembre 2016



Developing Enterprise Java Application with Hibernate

Procesamiento por Lotes

- Una mala idea para hacer importaciones de datos masivos en Hibernate podría ser esto:

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
    Customer customer = new Customer(.....);
    session.save(customer);
}
tx.commit();
session.close();
```

.es ©. Noviembre 2016

216

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Procesamiento por Lotes

- Este código puede llegar a producir una excepción de tipo:
OutOfMemoryException
- Hibernate **mantiene en caché** todas las instancias de Customer.
- Para realizar procesamientos por lotes es necesario habilitar el lote JDBC.
- Es importante para obtener un rendimiento óptimo.
hibernate.jdbc.batch_size 20
Debería estar dentro del rango 10 a 50.
- También se puede realizar este tipo de trabajo cuando la caché de 2º nivel esté desactivada.
hibernate.cache.use_second_level_cache false
- Estas propiedades se establecen en el fichero de configuración de Hibernate: **hibernate.cfg.xml**

.es ©. Noviembre 2016

217

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Inserciones por lotes

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

for ( int i=0; i<100000; i++ ) {
    Customer customer = new Customer("....");
    session.save(customer);

    if ( i % 20 == 0 ) {
        //20, same as the JDBC batch size
        //flush a batch of inserts and release memory:
        session.flush();
        session.clear();
    }
}
tx.commit();
session.close();
```

- Al hacer persistentes los objetos nuevos objetos es necesario utilizar los métodos:
- flush()** y luego **clear()** en la sesión de forma regular para controlar el tamaño de la caché de primer nivel.

.es ©. Noviembre 2016

218

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Actualizaciones de lotes

- Para recuperar y actualizar datos se aplican las mismas ideas.
- Además, necesita utilizar **scroll()** para sacar ventaja de los cursores del lado del servidor en consultas que retornen muchas filas de datos.

.es ©. Noviembre 2016 219

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Código

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
ScrollableResults customers = session.getNamedQuery("GetCustomers")
    .setCacheMode(CacheMode.IGNORE)
    .scroll(ScrollMode.FORWARD_ONLY);

int count=0;

while ( customers.next() ) {
    Customer customer = (Customer) customers.get(0);
    customer.updateStuff(...);

    if ( ++count % 20 == 0 ) {
        //flush a batch of updates and release memory:
        session.flush();
        session.clear();
    }
}
tx.commit();
session.close();
```

Para recuperar una consulta que Tiene muchos resultados. Se aconseja utilizar **ScrollableResults**

.es ©. Noviembre 2016 220

Developing Enterprise Java Application with Hibernate

La interfaz de Sesión sin estado

- Opcionalmente, Hibernate proporciona una **API orientada a comandos** que se puede utilizar para datos que concurren desde y hacia la base de datos en forma de **objetos separados**.
- Una **StatelessSession** no tiene un contexto de persistencia asociado con él.
- Y además:
 - No hay caché de primer nivel.
 - No hay caché de segundo nivel.
 - Se ignoran eventos e interceptores de Hibernate.
 - Se ignoran las colecciones.
- Una sesión sin estado es una abstracción en un nivel “más bajo”, mucho más cerca del JDBC subyacente.

.es ©. Noviembre 2016 221

Developing Enterprise Java Application with Hibernate

Código

```
StatelessSession session = sessionFactory.openStatelessSession();
Transaction tx = session.beginTransaction();

ScrollableResults customers =
    session.getNamedQuery("GetCustomers")
    .scroll(ScrollMode.FORWARD_ONLY);

while ( customers.next() )
{
    Customer customer = (Customer) customers.get(0);
    customer.updateStuff(...);
    session.update(customer);
}

tx.commit();
session.close();
```

Las instancias de customer NUNCA
 Se asocian con ningún contexto de
 Persistencia.

.es ©. Noviembre 2016 222

 **Developing Enterprise Java Application with Hibernate**

StatelessSession

- Define operaciones directas con las filas de la base de datos.
- **Son operaciones de ejecución inmediata:**
 - INSERT
 - UPDATE
 - DELETE
- Tienen una semántica diferente a las definidas por Session como son:
 - save(), saveOrUpdate() y delete().

.es ©. Noviembre 2016 223

 **Developing Enterprise Java Application with Hibernate**

HQL en Hibernate

.es ©. Noviembre 2016



Developing Enterprise Java Application with Hibernate

Hibernate Query Language

- Introducción.
 - Tipos de queries.
- HQL

.es ©. Noviembre 2016 225



Developing Enterprise Java Application with Hibernate

Tipos de Queries

- Para crear consultas en Hibernate disponemos de varios tipos de queries:
 - **HQL:** Hibernate Query Language.
 - Las consultas se realizan sobre los objetos mapeados.
 - **API de Criteria:**
 - En este caso las consultas se NO realizan utilizando HQL o SQL, en este caso se realizan estableciendo restricciones.
 - **QBE: Query By Example:**
 - Las condiciones se establecen en objetos asociados. Se pueden establecer filtros.
 - **SQL Nativo:**
 - Un método parecido al SQL.

.es ©. Noviembre 2016 226

 Developing Enterprise Java Application with Hibernate

HQL

- Hibernate Query Language.
- Es un lenguaje de consulta más simplificado que el SQL y que **se realiza sobre los objetos NO sobre las tablas**.
- Hibernate nos abstrae de las bases de datos y trabajamos a nivel de objeto.
- NO es sensible a mayúsculas y minúsculas. Salvo los nombres de las clases y las propiedades de Java.

.es ©. Noviembre 2016 227

 Developing Enterprise Java Application with Hibernate

HQL

- Ordenación de resultados
- Paginación de resultados
- Agregación mediante group by, having, y funciones agregadas: avg, sum, min, max,...
- Outer joins cuando se recuperan múltiple objetos por fila.
- La habilidad de llamar a funciones SQL estándar y definidas por el usuario.
- Subconsultas

.es ©. Noviembre 2016 228



HQL

- Las consultas en Hibernate se encuentran estructuradas de forma similar al tradicional SQL con las típicas cláusulas SELECT, FROM y WHERE.
- Una consulta simple podría ser:
“from Cliente” → equivale a:
Select * from clientes;
- Las consultas se realizan sobre los objetos que están mapeados.



HQL

- **Query:** Este interfaz permite crear consultas y enlazar argumentos a parámetros de la consulta (binding).
- Permite definir consultas en HQL (Hibernate Query Language) o en SQL.
- A partir de esta clase podemos obtener colecciones de objetos cuando ejecutamos una consulta.

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

- A partir de un objeto Trabajador que tenemos mapeado en Hibernate:


```
String sql = "from Trabajador";
Query query = session.createQuery(sql);
List trabajadores = query.list();
```
- Obtenemos todos los trabajadores de la tabla.
- También podemos utilizar listas parametrizadas:


```
List<Trabajador>.
```

.es ©. Noviembre 2016 231

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Login SQL

- De forma opcional podemos activar una propiedad dentro del fichero de configuración de Hibernate para que nos muestre las consultas de SQL que va lanzando.
- Dentro del fichero: hibernate.cfg.xml
 - Dentro de session-factory
 - <property name="hibernate.show_sql">true</property>

.es ©. Noviembre 2016 232

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

From

- En la cláusula from se pueden utilizar alias igual que en el SQL para luego nombrar propiedades de una clase u otra.
- Por convención se aconseja que los alias coincidan con el nombre de la clase y en minúsculas de la misma forma que nombramos objetos java.
- select c.nombre from Clientes c
 - Tenemos la opción de utilizar alias o no.

.es ©. Noviembre 2016 233

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Select

- Podemos consultar sólo por una propiedad de la clase.
- En ese caso:
 - select nombre from Clientes
 - select c.nombre from Clientes
 - Si lanzamos consultas del estilo:
 - nombre from Clientes
 - c.nombre from Clientes c
 - **Obtendremos ERROR!!**

.es ©. Noviembre 2016 234

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Select

- Cuando recuperamos más de una propiedad nos devuelve un: List<Object[]>, cada fila es un array de Object.
- Ejemplo:


```
String hql="select e.id, e.firstName from Employee e";
Query q = session.createQuery(hql);
List<Object[]> employees= (List<Object[]>)q.list();
for(Object[] employee: employees){
    Integer id = (Integer)employee[0];
    String firstName = (String)employee[1]; ....
}
```

.es ©. Noviembre 2016 235

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Select

- Sintaxis general:


```
Select distinct property, ..
From path [[AS] Alias] [FETCH ALL PROPERTIES]
Where LogicalExpression
Group By Property
Having logicalExpression
Order By property [ASC | DESC] ...
```

.es ©. Noviembre 2016 236

Developing Enterprise Java Application with Hibernate

Consultas con parámetros

```
// Obtener todos los Trabajadores de un determinado departamento:  
String sql = "from Trabajador where departamento = ?";  
Query query = session.createQuery(sql);  
query.setText(0, "compras");  
List<Trabajador> trabajadores = (List<Trabajador>)query.list();  
  
// Otra forma de hacerlo: Le podemos dar un nombre al parámetro.  
String sql2 = "from Trabajador where departamento = :departamento";  
Query query2 = session.createQuery(sql2);  
query2.setText("departamento", "compras");  
List<Trabajador> trabajadores2 = (List<Trabajador>)query2.list();
```

Práctica 9

.es ©. Noviembre 2016 237

Developing Enterprise Java Application with Hibernate

Restricciones HQL

- **OPERADORES:**
 - matemáticos: +, -, *, /
 - comparación binarios: =, >=, <=, <>, !=, like
 - lógicos: and, or, not
- Paréntesis () que indican agrupación
- in, not in, between, is null, is not null, is empty, is not empty, member of y not
- member of
- Caso "simple", case ... when ... then ... else ... end, y caso "buscado", case when ... then ... else ... End
- **FUNCIONES:** (*sopportadas por la Base de datos*)
 - concatenación de cadenas ... | ... o concat(...,...)
 - current_date(), current_time() y current_timestamp()
 - second(...), minute(...), hour(...), day(...), month(...), and year(...)
 - substring(), trim(), lower(),
 - upper(), length(), abs(), sqrt(), mod() ...

.es ©. Noviembre 2016 238

Developing Enterprise Java Application with Hibernate

Update

- Sintaxis:


```
UPDATE [VERSIONED]
[FROM] path [[AS] alias], []
SET Property= value [,...]
[WHERE logicalExpression]
```
- Ejemplo:


```
String hql="update Stock set stockName = :stockName" + "
           where stockCode = :stockCode";
Query query = session.createQuery(hql);
query.setParameter("stockName", "DIALOG1");
query.setParameter("stockCode", "7277");
int result = query.executeUpdate();
```

.es ©. Noviembre 2016 239

Developing Enterprise Java Application with Hibernate

Delete

- Sintaxis:


```
DELETE [FROM] path [[AS] alias]
[WHERE logicalExpression]
```
- Ejemplo:


```
String hql="delete Stock where stockCode = :stockCode";
Query query = session.createQuery(hql);
query.setParameter("stockCode", "7277");
int result = query.executeUpdate();
```

.es ©. Noviembre 2016 240

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Insert

- Sintaxis:

```
INSERT INTO path(property...) select ...
```
- Ejemplo:

```
Query query = session.createQuery("insert into
    Stock(stock_code, stock_name)" + "select
    stock_code, stock_name from backup_stock");
int result = query.executeUpdate();
```

.es ©. Noviembre 2016 241

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Resultado de las consultas (Paginación)

- Para recorrerlas podemos utilizar un bucle for.
- Podemos limitar el numero de resultados sobre la query.
 – **query.setMaxResults(int)**
- También podemos indicar en que registro queremos empezar
 → paginación de listados.
 – **query.setFirstResult(int);**
- Para poner condiciones en la query podemos utilizar preparedStatement.
- “from Trabajador where nombre = ?”

Práctica 3

.es ©. Noviembre 2016 242



Ejemplo con parámetros

- Lo podemos utilizar todas las veces que queramos.

```
Query q = session.createQuery("from Event where "+  
"startDate = :startDate or endDate < :startDate");  
q.setParameter("startDate", eventStartDate);  
List results = q.list();
```



Claúsulas HQL

- **FROM:**
- Esta cláusula te permite especificar los objetos que estás buscando.
- Permite crear **alias**.
 - Ejemplo:
 - from Event e where e.name='Opening Plenary'

ORDER BY

- La lista retornada por una consulta se puede ordenar por cualquier propiedad de una clase retornada o componentes:
- Ejemplo:
 - *from DomesticCat cat order by cat.name asc, cat.weight desc, cat.birthdate*
- Los **asc** o **desc** opcionales indican ordenamiento ascendente o descendente respectivamente.
- Por defecto hace ordenación ascendente si no indicamos nada.

Asociaciones

- También puede asignar alias a entidades asociadas o a elementos de una colección de valores utilizando una join.
- Por ejemplo:


```
from Cat as cat inner join cat.mate as mate
left outer join cat.kittens as kitten
```



```
from Cat as cat left join cat.mate.kittens as kittens
```

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Subconsultas

```
from Cat as fatcat
where fatcat.weight > (select avg(cat.weight) from DomesticCat cat)

from DomesticCat as cat
where cat.name = some (select name.nickName from Name as name)

from Cat as cat
where not exists (from Cat as mate where mate.mate = cat)

from DomesticCat as cat
where cat.name not in (select name.nickName from Name as name)
```

.es ©. Noviembre 2016 247

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Funciones de Agregado

- Son válidas consultas del tipo:
 - SELECT **COUNT(*)** FROM Persona
 - SELECT **AVG(P.id)** FROM Persona P
 - SELECT **SUM(importe)** FROM Pedido
 - SELECT **MAX(P.id)**, **MIN(P.id)** FROM Persona P

.es ©. Noviembre 2016 248



Definir escalares

```
String sql = "select avg(p.precio) as avgPrecio from productos p";
SQLQuery sqlQuery = session.createSQLQuery(sql);
sqlQuery.addScalar("avgPrecio");
Double d = (Double) sqlQuery.uniqueResult();
System.out.println("avgPrecio: " + d);
```

- **productos:** Representa el nombre de una tabla de la BD.
- Si vamos a obtener varios resultados:
 - List resul = sqlQuery.list();
 - System.out.println(resul);



Consultas polimórficas

- **from java.lang.Object**
 - Pasará por todas las instancias mapeadas.
 - Sería el equivalente a un select de todas las tablas.
 - Lista todas las tablas una detrás de otra.

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

SQL Nativo

- También disponemos de un método para lanzar consultas en SQL.
- El método se aplica sobre la session.
– **createSQLQuery(sql)**.
- Y se representa mediante el objeto **SQLQuery**.

.es ©. Noviembre 2016 251

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();

String sql = "select {Cliente.*} from clientes {Cliente}";
SQLQuery sqlQuery = session.createSQLQuery(sql);
sqlQuery.addEntity("Cliente", Cliente.class);
List<Cliente> clientes = sqlQuery.list();

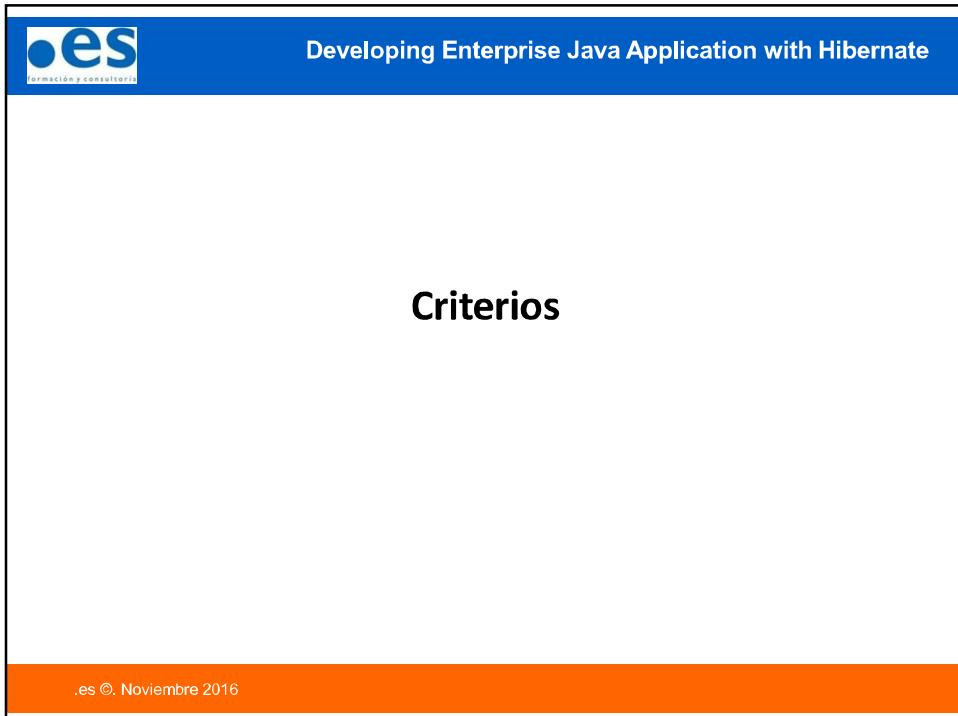
for (Cliente cliente : clientes)
    System.out.println(cliente);

session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
```

Indicar la clase con lo que se va a llenar el resultado.

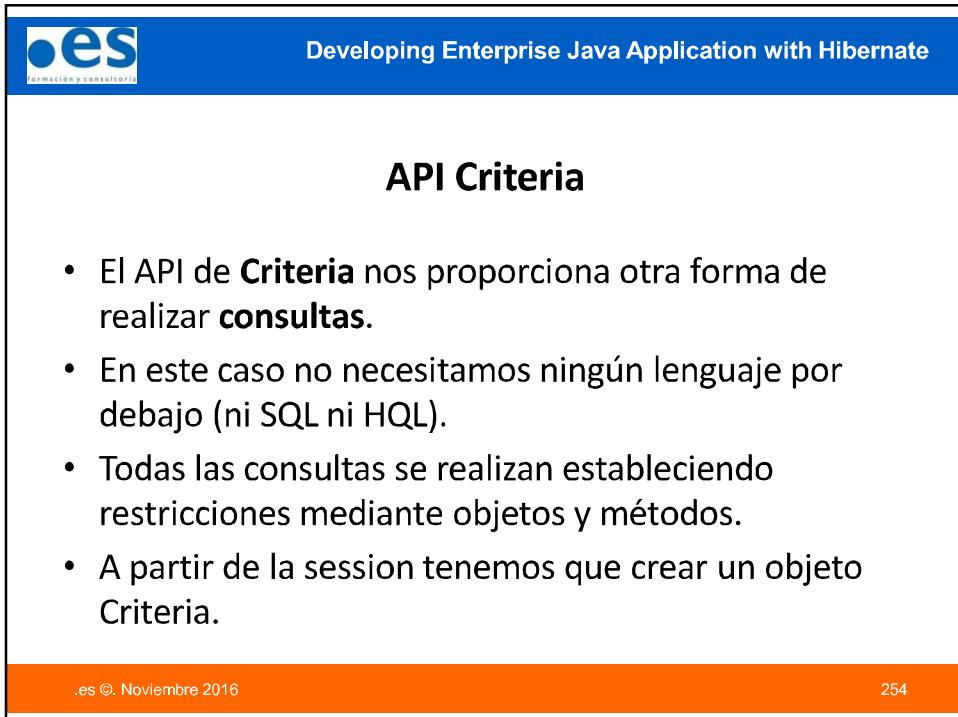
Práctica 8

.es ©. Noviembre 2016 252



The image shows a slide from a course titled "Developing Enterprise Java Application with Hibernate" by ".es formación y consultoría". The slide has a blue header bar with the logo and title. The main content area is white and contains the section title "Criterios". At the bottom of the slide is an orange footer bar with the text ".es ©. Noviembre 2016".

Criterios



The image shows a slide from a course titled "Developing Enterprise Java Application with Hibernate" by ".es formación y consultoría". The slide has a blue header bar with the logo and title. The main content area is white and contains the section title "API Criteria". Below it is a bulleted list of five points explaining the benefits of the Criteria API:

- El API de **Criteria** nos proporciona otra forma de realizar **consultas**.
- En este caso no necesitamos ningún lenguaje por debajo (ni SQL ni HQL).
- Todas las consultas se realizan estableciendo restricciones mediante objetos y métodos.
- A partir de la session tenemos que crear un objeto Criteria.

.es ©. Noviembre 2016 254

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

API Criteria

- Disponemos de la clase **Criteria** en org.hibernate.
- Para crear un objeto de estos lo hacemos a partir de la session:
 - Indicando la clase sobre la que queremos realizar consultas.
 - Ejemplo:

```
Criteria criterios = session.createCriteria(MiClase.class)
```

.es ©. Noviembre 2016 255

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Obtener resultados

```
Criteria crit = session.createCriteria(Cat.class);
crit.setMaxResults(50);
crit.setFirstResult(10);
List cats = crit.list();
```

- Al igual que con query podemos obtener los resultados mediante una colección.
- Podemos paginar los resultados.

.es ©. Noviembre 2016 256

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Limitar los resultados

- Disponemos de la clase **Restrictions** para añadir limitaciones a los resultados, sería el equivalente a **where** en el SQL.
- Podemos establecer filtros de tipo **like**, **between**, **and**, **or**, etc.
- Los criterios siempre hacen referencia al nombre de las propiedades de los objetos.
- Cuando queremos agregar restricciones utilizamos el método **add** de la clase Criteria.
 - crit.add(Restrictions.like(...));

.es ©. Noviembre 2016

257

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Restricción like

- Para utilizar like:
 - **Restrictions.like("nombre", "cadena");**
 - Indicamos el nombre de la propiedad de la clase y la cadena formato de like.
 - Ejemplo:
 - Criteria crit = session.createCriteria(Persona.class);
 - crit.**add**(Restrictions.like("nombre", "A%"))
 - List resultados = crit.list();
 - La lista de los nombres que empiezan por 'A'.

.es ©. Noviembre 2016

258

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Restricción between

- Para utilizar between:
 - Restrictions.between("nombre", min, max);
 - Indicamos el nombre del atributo y los rangos mínimos y máximos.

- Ejemplo:
 - Criteria crit = session.createCriteria(Persona.class);
 - crit.add(Restrictions.between ("edad", new Integer(20), new Integer(30)));
 - List resultados = crit.list();
 - Resultados entre 20 y 30.

.es ©. Noviembre 2016 259

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Restricción or

- Podemos establecer criterios **or**:


```
crit.add(Restrictions.or(
    Restrictions.like("nombre", "A%"),
    Restrictions.between("edad", new Integer(20), new
    Integer(30))));
```

 - El resultado será las personas que su nombre empieza por A o su edad está entre 20 y 30.

.es ©. Noviembre 2016 260

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Implementar el and

- Sería agregando las restricciones por separado, utilizando dos veces el add (o las que se necesiten):
- Ejemplo:


```
crit.add(Restrictions.like("nombre", "A%"));
crit.add(Restrictions.between("edad", new
    Integer(20), new Integer(30)));
```

.es ©. Noviembre 2016 261

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Mas restricciones

- **equals:** Restrictions.eq("nombre", valor);
- **Mayor que:** Restrictions.gt("nombre",valor);
- **Menor que:** Restrictions.lt("nombre",valor);
- **is null:** Restrictions.isNull("nombre");
- **in:** Restrictions.in("nombre", String []valores)
 - Analiza si el atributo se encuentra dentro de un conjunto de valores, pasado como un array de String.

.es ©. Noviembre 2016 262

 Developing Enterprise Java Application with Hibernate

Ordenación de Resultados

- Los resultados se pueden ordenar mediante: `org.hibernate.criterion.Order`.
- Se pueden establecer **asc / desc** y por varios campos.
- Ejemplo:
 - Indicamos asc / desc mediante un método y el nombre del atributo por el que queremos ordenar.
 - Podemos añadir varias ordenaciones.
 - `criterios.addOrder(Order.asc("pais"));`

.es ©. Noviembre 2016 263

 Developing Enterprise Java Application with Hibernate

Obtener un único resultado

- Criteria dispone del método **uniqueResult()** que devuelve una única instancia o null en caso de no encontrarla.
- Lo devuelve como Object, así que hay que hacer un casting a nuestra clase.
- Por ejemplo, partiendo de una clase como Persona con id y nombre como atributos.

.es ©. Noviembre 2016

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

- Si ya tenemos creada una session, haremos:

```
Criteria criteria = session.createCriteria(Persona.class);
criteria.add(Restrictions.eq("id", new Long(1)));
Object result = criteria.uniqueResult();
if (result != null) {
    Persona persona = (Persona) result;
    System.out.println("Persona = " + persona.getName());
}
```

.es ©. Noviembre 2016

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

QBE: Query By Example

- QBE:** Nos permite establecer filtros en objetos asociados, es decir, cuando tenemos propiedades que son otros objetos.
- Por ejemplo, suponemos que tenemos el objeto Pedido y queremos establecer un filtro por el nombre del cliente.
- El objeto Pedido, tiene una propiedad que es el cliente.

.es ©. Noviembre 2016

266

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

```
// Podemos utilizar un objeto como Filtro, se le configuran las propiedades que
// queramos y el resto se ignoran:
Criteria criterios = session.createCriteria(Cliente.class);

// Clientes de Alemania:
Cliente cliente = new Cliente();
cliente.setPais("Alemania");
criterios.add(Example.create(cliente));

List<Cliente> clientes = (List<Cliente>)criterios.list();
```

.es ©. Noviembre 2016 267

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

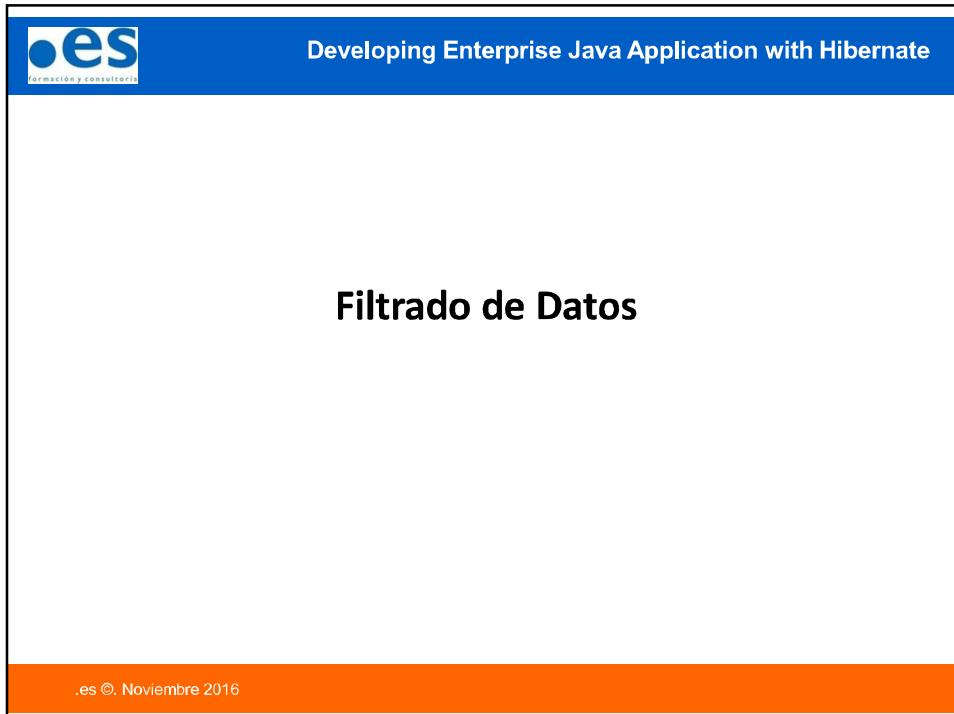
```
// Creamos el objeto criteria a partir de la Session y asociado a la clase Pedido:
Criteria criteria = session.createCriteria(Pedido.class);

// Cargo el cliente:
cliente = (Cliente)session.get(Cliente.class, id_cliente);

// Establecer el criterio sobre la propiedad cliente.
• criteria.createCriteria("cliente").add(Example.create( cliente));
```

Práctica 7

.es ©. Noviembre 2016 268



The image shows a slide from a course titled "Developing Enterprise Java Application with Hibernate" by ".es Formación y consultoría". The slide has a blue header bar with the logo and title. The main content area is white and contains the section title "Filtrado de Datos". At the bottom of the slide is an orange footer bar with the text ".es ©. Noviembre 2016".

Filtrado de Datos

.es ©. Noviembre 2016



The image shows a slide from a course titled "Developing Enterprise Java Application with Hibernate" by ".es Formación y consultoría". The slide has a blue header bar with the logo and title. The main content area is white and contains a section titled "Filtros" followed by a bulleted list of points. At the bottom of the slide is an orange footer bar with the text ".es ©. Noviembre 2016" and the number "270".

Filtros

- Hibernate permite la definición de filtros en tiempo de compilación.
- Se pueden establecer en los archivos de mapeo XML o con anotaciones en las clases.
- Es una forma de obtener resultados, más administrable y parametrizable. Aunque esto mismo se puede hacer con HQL.
- Nos pueden evitar tener que reescribir las consultas de acceso a los datos que aparecen a lo largo de la aplicación, estableciendo simplemente los filtros necesarios cuando se crea la sesión.

.es ©. Noviembre 2016 270



Developing Enterprise Java Application with Hibernate

Filtros

- Un filtro se puede activar o desactivar según nos interese.
- Cuando lancemos una query si el filtro está activado actuará si no obtendremos todos los resultados.

.es ©. Noviembre 2016 271



Developing Enterprise Java Application with Hibernate

Filtros XML

- A partir de la tabla:

```
CREATE TABLE IF NOT EXISTS empleados (
    usuario  VARCHAR(25) NOT NULL,
    nombre   VARCHAR(128),
    genero   VARCHAR(1),
    PRIMARY KEY (usuario)
) ENGINE=InnoDB CHARSET=utf8 collate=utf8_general_ci;
```

.es ©. Noviembre 2016 272

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Filtros XML

- En el archivo de mapeo: Empleados.hbm.xml

```
<hibernate-mapping>
<class name="dao.Empleado" table="empleados" >
<id name="usuario" type="string">
<generator class="assigned"/>
</id>
<property name="nombre" type="string" length="128"/>
<property name="genero" type="string" length="1"/>

<filter name="filtroGenero" condition=":generoParam = genero"></filter>
</class>

<filter-def name="filtroGenero">
<filter-param name="generoParam" type="string"/>
</filter-def>
```

</hibernate-mapping>

El filtro se declara fuera de la clase.
La condición dentro de la clase
El filtro se activa por código cuando nos interese.

.es ©. Noviembre 2016

273

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

En el código

- Creamos la sessionFactory
- Abrir una session
- // Habilitar el filtro:
- Filter filtro = session.enableFilter("filtroGenero");
- filtro.setParameter("generoParam", "F");
- Lanzar una query, normal “FROM Empleado”. El filtro actuará sólo recuperando los empleados que cumplen la condición.
- // Deshabilitar el filtro:
- Session.disabledFilter("filtroGenero");
- Cerrar la session.

.es ©. Noviembre 2016

274

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Filtros con Anotaciones

- Disponemos de dos anotaciones equivalentes a las etiquetas de XML. La conexión entre la definición del filtro (@FilterDef y @Filter) se realiza a través del **nombre**:
- Con @FilterDef definimos el filtro, el nombre y los parámetros.
- Con @Filter se indica el nombre para ligarlo a la definición anterior y se indica la condición.

```
@Entity
@FilterDef(name="minLength",
    parameters=@ParamDef( name="minLength", type="integer" ) )
@Filter(name="minLength", condition=":minLength <= length")
public class Forest { ... }
```

.es ©. Noviembre 2016 275

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Anotaciones

- **@FilterDef**
 - @FilterDef(name="<nombre>",
• parameters=@ParamDef(name="<nombre>" type="<type>")
 - Cuando tengamos mas de un parámetro,
• Parameters={@ParamDef(...), @ParamDef(...)}
- **@ParamDef**
 - Definir parámetros en un filtro.
- **@FilterDefs**
 - Cuando tenemos definir más de un filtro (@FilterDef)


```
@FilterDefs({
    @FilterDef(...),
    @FilterDef(...),
    @FilterDef(...),
})
```

.es ©. Noviembre 2016 276



Developing Enterprise Java Application with Hibernate

Anotaciones

- **@Filter**
 - Para definir la condición de un filtro:
`@Filter(name="<name>", condition="<condition>")`
 - Dentro de las condiciones hacemos referencia a las propiedades del bean y a los parámetros que tienen que ir precedidos de ":".
 - :miparam
- **@Filters**
 - Se utiliza cuando tenemos que definir varios Filter
`@Filters({
 @Filter(...),
 @Filter(...)
})`

.es ©. Noviembre 2016 277



Developing Enterprise Java Application with Hibernate

Hibernate Tools Eclipse / STS

.es ©. Noviembre 2016

Developing Enterprise Java Application with Hibernate

Hibernate Tools

- Herramientas que nos facilitan el desarrollo con Hibernate. Dando la posibilidad de crear los archivos de mapeo y de configuración de forma automatizada.
- Tenemos un plugin que se puede instalar dentro de eclipse.
- Se instalan desde eclipse /STS .
- Forman parte de las **JBOSS Tools**

.es ©. Noviembre 2016 279

Developing Enterprise Java Application with Hibernate

Instalación

- Desde: menú help → eclipse **MarketPlace**.
- Buscar “Jboss tools”,



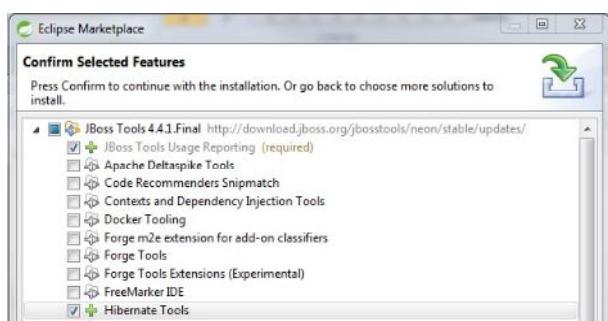
The screenshot shows the Eclipse Marketplace interface. A search bar at the top contains the text "jboss tools". Below it, a list item for "JBoss Tools 4.4.1.Final" is highlighted. The description for this item states: "JBoss Tools is an umbrella project for a set of Eclipse plugins that includes support for JBoss and related technologies, such as Hibernate, JBoss AS, CDI, ... more info". It also mentions "by Red Hat, Inc., EPL" and "jbosstools maven hibernate Mobile JSF ...". At the bottom right of the list item, there is a button labeled "Install Pending".

.es ©. Noviembre 2016 280

Developing Enterprise Java Application with Hibernate

Instalación (2^a paso)

- Después seleccionar la casilla de **Hibernate Tools**.



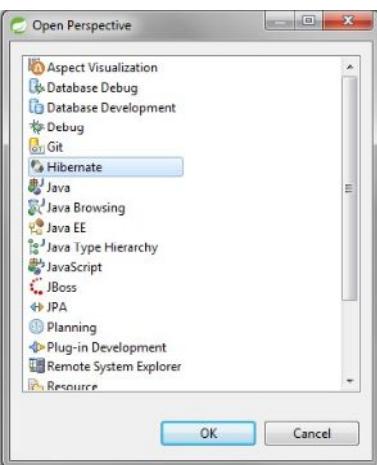
.es ©. Noviembre 2016 281

Developing Enterprise Java Application with Hibernate

Comprobar si está instalado

ABRIR LA PERSPECTIVA DE HIBERNATE

- Menú Window
- Perspective
- Open Perspective



.es ©. Noviembre 2016 282

Developing Enterprise Java Application with Hibernate

Instalación

- La instalación se puede ejecutar en modo background, mientras se descarga.
- Una vez que esté instalado, creamos un proyecto java.
- Después con el botón derecho:
 - New → Other → Hibernate
 - Nos aparecen estas 4 opciones:

- <http://www.mkyong.com/hibernate/how-to-generate-code-with-hibernate-tools/>

.es ©. Noviembre 2016 283

Developing Enterprise Java Application with Hibernate

Crear un proyecto Maven

.es ©. Noviembre 2016 284

Developing Enterprise Java Application with Hibernate

Dependencias Maven

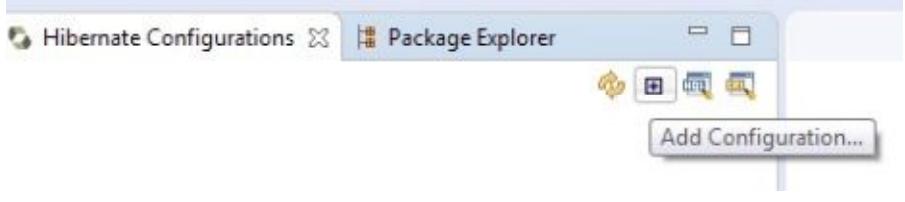
En el POM.XML

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>4.3.11.Final</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.23</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.3.1.Final</version>
</dependency>
```

.es ©. Noviembre 2016 285

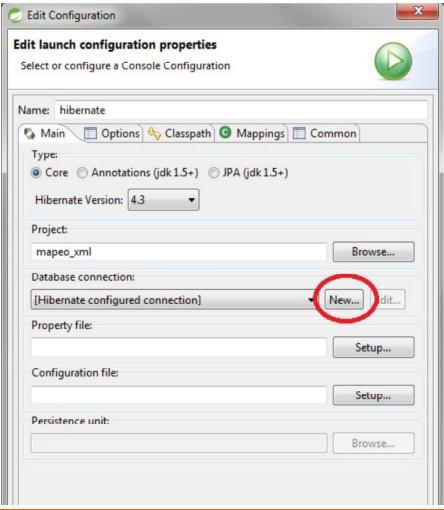
Developing Enterprise Java Application with Hibernate

Perspectiva Hibernate



.es ©. Noviembre 2016 286

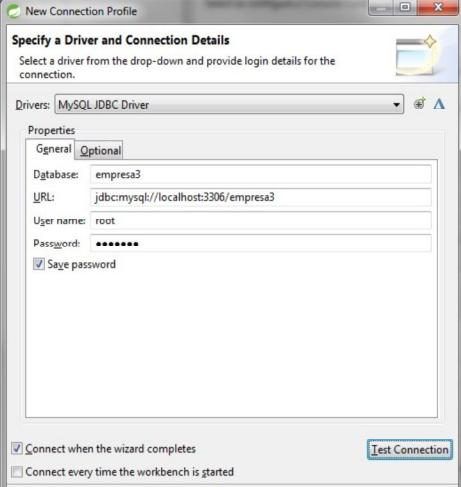
Perspectiva Hibernate



- Al crear la conexión seleccionar la BD.
- El driver es necesario (carpeta software)
- Datos de la conexión y **“Test Connection”**
- Ok, finish

.es ©. Noviembre 2016 287

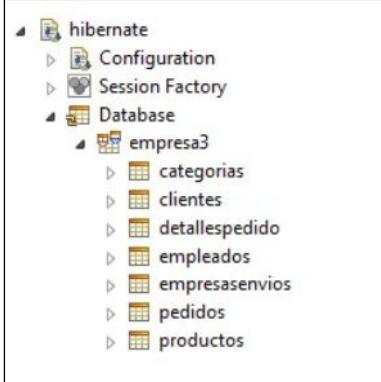
Datos conexión



.es ©. Noviembre 2016 288

Developing Enterprise Java Application with Hibernate

Edit configuration

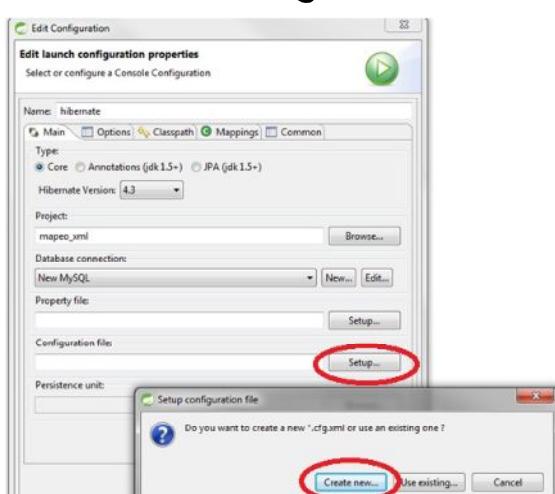


- Botón derecho sobre “Hibernate”
- Edit configuration

.es ©. Noviembre 2016 289

Developing Enterprise Java Application with Hibernate

Edit Configuration



.es ©. Noviembre 2016 290

Developing Enterprise Java Application with Hibernate

Edit Configuration

.es ©. Noviembre 2016 291

Developing Enterprise Java Application with Hibernate

Hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">org.gjt.mm.mysql.Driver</property>
    <property name="hibernate.connection.password">antonio</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/empresa3</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  </session-factory>
</hibernate-configuration>
```

.es ©. Noviembre 2016 292

Developing Enterprise Java Application with Hibernate

Generar código I (con mapeo XML)

The screenshot shows the 'Hibernate Code Generation Configurations' dialog. The 'Main' tab is selected. The configuration is named 'New_configuration'. The 'Console configuration' dropdown is set to 'hibernate'. The 'Output directory' is set to '\mapeo_xml\src\main\java'. The 'Package' is 'com.oracle.pojo'. Under 'reveng.xml:', the checkbox 'Reverse engineer from JDBC Connection' is checked. In the 'reveng.strategy:' section, several checkboxes are checked: 'Generate basic typed composite ids', 'Detect optimistic lock columns', 'Detect many-to-many tables', and 'Detect one-to-one associations'. The 'Template directory' dropdown has three options: 'Template...', 'Filesystem...', and 'Workspace...'. At the bottom right of the dialog are 'Template...', 'Filesystem...', and 'Workspace...' buttons.

.es ©. Noviembre 2016 293

Developing Enterprise Java Application with Hibernate

Generar código II (con Mapeo XML)

The screenshot shows the 'Hibernate Code Generation Configurations' dialog. The 'Exporters' tab is selected. The configuration is named 'New_configuration'. Under 'General settings', the checkbox 'Generate EJB3 annotations' is unchecked. In the 'Exporters' list, several checkboxes are checked: 'Domain code (java)', 'Hibernate XML Mappings (.hbm.xml)', 'DAO code (java)', 'Entity Relationship (ER) intermediate', and 'Hibernate XML Configuration (.cfg.xml)'. The 'Properties' table is empty. At the bottom right of the dialog are 'Revert', 'Apply', and 'Run' buttons. To the right of the dialog, a file tree shows the project structure: 'mapeo_xml' containing 'src/main/java' (with 'App.java' and 'com.oracle.pojo' package), 'src/test/java', 'JRE System Library [J2SE-1.5]', 'Maven Dependencies', 'src', 'target', and 'pom.xml'. The 'Run' button is highlighted with a red box.

.es ©. Noviembre 2016 294

Developing Enterprise Java Application with Hibernate

Hibernate.cfg.xml (con XML)

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.bytecode.use_reflection_optimizer">false</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.password">antonio</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/empresa3</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.search.autoregister_listeners">true</property>
    <property name="hibernate.validator.aply_to_ddl">false</property>
    <mapping resource="com/oracle/pojo/Categorias.hbm.xml" />
    <mapping resource="com/oracle/pojo/Pedidos.hbm.xml" />
    <mapping resource="com/oracle/pojo/Empleados.hbm.xml" />
    <mapping resource="com/oracle/pojo/Productos.hbm.xml" />
    <mapping resource="com/oracle/pojo/Detallespedido.hbm.xml" />
    <mapping resource="com/oracle/pojo/Empresasenvios.hbm.xml" />
    <mapping resource="com/oracle/pojo/Clientes.hbm.xml" />
  </session-factory>
</hibernate-configuration>
```

.es ©. Noviembre 2016 295

Developing Enterprise Java Application with Hibernate

Generar código (Anotaciones)

.es ©. Noviembre 2016 296

Developing Enterprise Java Application with Hibernate

Hibernate.cfg.xml (con Anotaciones)

```

<hibernate-configuration>
    <session-factory>
        <property name="hibernate.bytecode.use_reflection_optimizer">false</property>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.password">antonio</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/empresa3</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.search.autoregister_listeners">true</property>
        <property name="hibernate.validator.apply_to_ddl">false</property>
        <mapping class="com.oracle.pojo.Categorias" />
        <mapping class="com.oracle.pojo.Pedidos" />
        <mapping class="com.oracle.pojo.Empleados" />
        <mapping class="com.oracle.pojo.Productos" />
        <mapping class="com.oracle.pojo.Detallespedido" />
        <mapping class="com.oracle.pojo.Empresasenvios" />
        <mapping class="com.oracle.pojo.Clientes" />
    </session-factory>
    ...

```

.es ©. Noviembre 2016 297

Developing Enterprise Java Application with Hibernate

Hibernate con Netbeans

.es ©. Noviembre 2016

Developing Enterprise Java Application with Hibernate

Crear la conexión

- Dentro de la pestaña de Services → Databases.
- Botón derecho → New Connection.

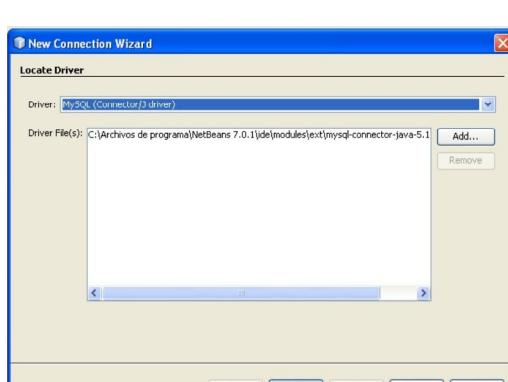


.es ©. Noviembre 2016 299

Developing Enterprise Java Application with Hibernate

Crear la conexión

- Si no tenemos el driver, pulsar Add para agregar el Driver de MySQL.



.es ©. Noviembre 2016 300

Developing Enterprise Java Application with Hibernate

Crear la conexión

- Especificar los parámetros de la conexión.

.es ©. Noviembre 2016

301

Developing Enterprise Java Application with Hibernate

Crear la conexión

- Una vez que hemos conectado podemos abrir la conexión y ver las tablas y los datos de la BD.

.es ©. Noviembre 2016

302

Developing Enterprise Java Application with Hibernate

Crear el proyecto

- Una vez que tenemos la conexión registrada en Netbeans.
- Creamos un proyecto: Java o Java Web
- Tenemos que marcar en el apartado de FrameWorks: Hibernate.
- Tenemos que seleccionar la conexión.

.es ©. Noviembre 2016 303

Developing Enterprise Java Application with Hibernate

Hibernate

- NetBeans nos crea un fichero XML con los datos de la conexión.
- Se ubica en el paquete default.
hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prop
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/empresa3<
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">antonio</property>
    </session-factory>
</hibernate-configuration>
```

.es ©. Noviembre 2016 304

Developing Enterprise Java Application with Hibernate

Hibernate

- Una vez creado el fichero de configuración, vamos a crear un fichero de ingeniería inversa a partir del esquema de la BD.
- Esto lo necesitamos para crear el modelo de objetos y los ficheros de mapeo.

.es ©. Noviembre 2016 305

Developing Enterprise Java Application with Hibernate

Hibernate

- Seleccionamos las tablas que queremos mapear:

.es ©. Noviembre 2016 306

Developing Enterprise Java Application with Hibernate

Hibernate

- Después de seleccionar las tablas, pulsar el botón de finish.
- Nos genera el fichero: hibernate.reveng.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-reverse-engineering PUBLIC
<schema-selection match-catalog="empresa3"/>
<table-filter match-name="clientes"/>
<table-filter match-name="productos"/>
<table-filter match-name="detallespedido"/>
<table-filter match-name="categorias"/>
<table-filter match-name="empresazenvios"/>
<table-filter match-name="empleados"/>
<table-filter match-name="pedidos"/>
<table-filter match-name="usuarios"/>
</hibernate-reverse-engineering>
```

.es ©. Noviembre 2016 307

Developing Enterprise Java Application with Hibernate

Hibernate

- Para crear los ficheros de mapeo y POJO (las clases java que representan las tablas de la BD).

Seleccionamos:

.es ©. Noviembre 2016 308

Developing Enterprise Java Application with Hibernate

Hibernate (generar ficheros XML)

- Seleccionamos el fichero de configuración y el fichero de ingeniería inversa.
- Marcar generar código JDK 5.

Con esta opción
Generamos los archivos
De mapeo XML (y las clases)

.es ©. Noviembre 2016 309

Developing Enterprise Java Application with Hibernate

Hibernate (generar Anotaciones)

- Con esta opción en vez de generar fichero XML de mapeo, generamos las clases a partir de la base de datos con **anotaciones**.

```

@Entity
@Table(name="productos"
      ,catalog="empresa3"
)
public class Productos implements java.io.Serializable {
}

```

.es ©. Noviembre 2016 310

Developing Enterprise Java Application with Hibernate

Hibernate

- Por cada clase crea un fichero de mapeo XML donde indica las propiedades de la tabla, indicando tipos, nombres de columnas, claves, etc.
- La tabla de categorías:

```
<hibernate-mapping>
    <class name="modelo.pojo.Categorias" table="categorias" catalog="empresa3">
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="identity" />
        </id>
        <property name="nombre" type="string">
            <column name="nombre" length="50" />
        </property>
        <set name="productoses" inverse="true">
            <key>
                <column name="idcategoria" not-null="true" />
            </key>
            <one-to-many class="modelo.pojo.Productos" />
        </set>
    </class>
</hibernate-mapping>
```

.es ©. Noviembre 2016 311

Developing Enterprise Java Application with Hibernate

Hibernate

- El siguiente paso es crear la clase HibernateUtil, será la clase principal para poder conectar con la BD y poder grabar y recuperar datos en la BD.
- Indicamos un nombre y un paquete donde se ubicará la clase.

.es ©. Noviembre 2016 312

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Hibernate

```
public static void main(String[] args) {
    SessionFactory factory;
    Session session;

    factory = NewHibernateUtil.getSessionFactory();
    session = factory.openSession();
    Query query = session.createQuery("FROM Productos");
    List<Productos> lista = query.list();

    for (Productos p : lista)
        System.out.println(p.getNombre());

}
```

.es ©. Noviembre 2016 313

•es
formación y consultoría

Developing Enterprise Java Application with Hibernate

Clase HibernateUtil 4.3

```
public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
        try {
            Configuration configuracion = new Configuration().configure();
            StandardServiceRegistryBuilder builder = new
            StandardServiceRegistryBuilder().applySettings(configuracion.getProperties());
            sessionFactory = configuracion.buildSessionFactory(builder.build());

        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

.es ©. Noviembre 2016 314

The image shows the front cover of a book. At the top, there is a blue header bar with the logo '.es Formación y consultoría' on the left and the title 'Developing Enterprise Java Application with Hibernate' in white text on the right. The main title 'Integración Spring y Hibernate' is centered below the header. At the bottom of the cover, there is an orange footer bar with the text '.es ©. Noviembre 2016'.

The image shows a page from the book's table of contents. It features the same blue header bar with the '.es' logo and title as the cover. Below the header, the word 'Contenido' is centered in a large, bold black font. A bulleted list follows, containing five items: 'HibernateTemplate.', 'HibernateDaoSupport.', 'Transacciones Spring para Hibernate.', and 'Integración.'. At the bottom of the page is an orange footer bar with the text '.es ©. Noviembre 2016' on the left and the page number '316' on the right.

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

HibernateTemplate

- Esta clase equivale a la clase JDBCTemplate para operaciones con la BD.
- **HibernateTemplate** nos proporciona los siguientes métodos, por ejemplo:
 - save(Object o)
 - update(Object o)
 - delete(Object o)
 - find(String sql)
 - get(Class class, Serializable id)
- **Más info:** <http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/orm/hibernate3/HibernateTemplate.html>

.es ©. Noviembre 2016 317

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

HibernateDaoSupport

- Para implementar el patrón DAO disponemos de esta clase.
- Tiene un método que nos proporciona una plantilla: `getHibernateTemplate()`.
- A partir de esta realizamos todas las operaciones: crear, borrar, actualizar, buscar, etc.

.es ©. Noviembre 2016 318

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

HibernateDaoSupport

- Esta clase tiene una propiedad: **sessionFactory** que en el contexto habrá que declarar un bean de tipo **LocalSessionFactoryBean** e injectarlo.
- Esto es muy similar a la clase **JdbcDaoSupport**, también disponía de un método **getJdbcTemplate** para realizar todas las operaciones y también había que injectar la propiedad **dataSource**.

.es ©. Noviembre 2016

319

•es
Formación y consultoría

Developing Enterprise Java Application with Hibernate

Ejemplo

- Definimos un **interface** con las operaciones:

```
public interface ClienteDao {  
    public void create(Cliente cliente);  
    public void delete(Cliente cliente);  
}
```

- La **clase** quedará:

```
public class ClienteDaoImpl extends  
    HibernateDaoSupport implements ClienteDao {}
```

.es ©. Noviembre 2016

320

Developing Enterprise Java Application with Hibernate

Transacciones

- Para realizar transacciones en Spring con Hibernate utilizaremos la clase:
 - HibernateTransactionManager.
 - Tendremos que definir un bean en el contexto de Spring.
 - Se configura con un sessionFactory (el equivalente a DataSource de JDBC).

```
<bean id="transactionManager"
      class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

.es ©. Noviembre 2016 321

Developing Enterprise Java Application with Hibernate

Integración

```

graph TD
    A[Aplicación] <--> B[Spring - BO]
    B <--> C[Hibernate - DAO]
    C <--> D[BD]
  
```

El diagrama muestra una jerarquía de integración entre cuatro componentes:

- Aplicación**: Se apoya en BO para realizar operaciones con la BD.
- Spring - BO**: Declaración de beans: dao, bo, sessionFactory, dataSource.
- Hibernate - DAO**: Archivos de mapeo: Beans ↔ Tablas de la BD.
- BD**: Representa la base de datos.

.es ©. Noviembre 2016 322

•es **Developing Enterprise Java Application with Hibernate**

Ejemplo (Integración con XML)

- Partimos de la siguiente tabla en la Base de datos:

.es ©. Noviembre 2016 323

•es **Developing Enterprise Java Application with Hibernate**

El proyecto

- Necesitamos las librerías de Spring, de Hibernate y el driver de MySQL:
- El paquete de recursos almacena los ficheros de configuración.
- El fichero de configuración de Spring:
 - applicationContext.xml se divide en:
 - Hibernate.xml**: Declaración de sessionFactory de Hibernate es gestionada por Spring.
 - Stock.xml**: Define beans para StockBo y StockDao.
 - dataSource.xml** : El dataSource con las propiedades de conexión especificadas en el fichero database.properties.

.es ©. Noviembre 2016 324

Developing Enterprise Java Application with Hibernate

Clase Stock / Archivo de Mapeo

```

public class Stock implements Serializable {

    private Integer stockId;
    private String stockCode;
    private String stockName;

    // Contructor, Get / Set, toString

<hibernate-mapping>
    <class name="com.curso.modelo.beans.Stock" table="stock" catalog="ejemplos">
        <id name="stockId" type="java.lang.Integer">
            <column name="STOCK_ID" />
            <generator class="identity" />
        </id>
        <property name="stockCode" type="string">
            <column name="STOCK_CODE" length="10" not-null="true" unique="true" />
        </property>
        <property name="stockName" type="string">
            <column name="STOCK_NAME" length="20" not-null="true" unique="true" />
        </property>
    </class>
</hibernate-mapping>

```

.es © Noviembre 2016 325

Developing Enterprise Java Application with Hibernate

Lógica de Negocio (BO)

```

public interface StockBo {
    public void save(Stock stock);
    public void update(Stock stock);
    public void delete(Stock stock);
    public List<Stock> selectAll();
    public Stock findByStockCode(String stockCode);
}

```

```

public class StockBoImpl implements StockBo {

    private StockDao stockDao;

    public StockDao getStockDao() {
        return stockDao;
    }
    public void setStockDao(StockDao stockDao) {
        this.stockDao = stockDao;
    }
    @Override
    public void save(Stock stock) {
        stockDao.save(stock);
    }
}

```

Get / Set de stockDao para poder injectar en el contexto de Spring.

Los otros métodos: save, delete, etc. Delegan al Dao (siguiente capa) las operaciones.

.es © Noviembre 2016 326

Developing Enterprise Java Application with Hibernate

Acceso a Datos (DAO)

```

public interface StockDao {
    public void save(Stock stock);
    public void update(Stock stock);
    public void delete(Stock stock);
    public List<Stock> selectAll();
    public Stock findByStockCode(String stockCode);
}

public class StockDaoImpl
extends HibernateDaoSupport
implements StockDao {

    @Override
    public void save(Stock stock) {
        getHibernateTemplate().save(stock);
    }

    @Override
    public void update(Stock stock) {
        getHibernateTemplate().update(stock);
    }
}

```

Todas las operaciones del Dao se realizan a partir de la plantilla: **HibernateTemplate**.

.es ©. Noviembre 2016 327

Developing Enterprise Java Application with Hibernate

Aplicación

```

ApplicationContext contexto = new ClassPathXmlApplicationContext("recursos/applicationContext.xml");
StockBo stockBo = (StockBo) contexto.getBean("stockBo");

```

En la aplicación, cargamos el contexto de Spring, recuperamos el Bean que representa la lógica de negocio (StockBo) y realizamos operaciones con él.

.es ©. Noviembre 2016 328

• El contexto de Spring se divide en varios ficheros (mejor organización):

```
<import resource="Hibernate.xml" />
<import resource="Stock.xml" />
<import resource="datasource.xml" />
```

• El fichero de propiedades: **database.properties (parámetros de conexión):**

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ejemplos
jdbc.username=root
jdbc.password=password
```

.es ©. Noviembre 2016 329

Hibernate.xml

```
<!-- Hibernate session factory -->
<bean id="sessionFactory"
      class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource">
      <ref bean="dataSource"/>
    </property>
    <property name="hibernateProperties">
      <props>
        <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
        <prop key="hibernate.show_sql">true</prop>
      </props>
    </property>
    <property name="mappingResources">
      <list>
        <value>recursos/Stock.hbm.xml</value>
      </list>
    </property>
  </bean>
```

sessionFactory se inicializan 3 propiedades:

- **dataSource**: un dataSource de Spring.
- **hibernateProperties**: El dialecto indica la BD con la que trabajamos y si queremos ver o no las sentencias SQL que genera Hibernate.
- **mappingResources**: lista con todos los archivos de mapeo de Hibernate.

.es ©. Noviembre 2016 330

Developing Enterprise Java Application with Hibernate

DataSource.xml

```
<bean
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location">
        <value>recursos/database.properties</value>
    </property>
</bean>

<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
</bean>

</beans>
```

Las propiedades se especifican con respecto a las claves definidas en el fichero: database.properties.

.es ©. Noviembre 2016 331

Developing Enterprise Java Application with Hibernate

Stock.xml

```
<!-- Stock business object -->
<bean id="stockBo" class="com.curso.modelo.bo.StockBoImpl" >
    <property name="stockDao" ref="stockDao" />
</bean>

<!-- Stock Data Access Object -->
<bean id="stockDao" class="com.curso.modelo.dao.StockDaoImpl" >
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>
```

Se definen los objetos dao y lógica de negocio.

.es ©. Noviembre 2016 332