MEDAC
Instituto Oficial de Formación Profesional

ACCESO A DATOS

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

El mapeo objeto relacional

07

ÍNDICE

/ 1. Introducción y contextualización práctica 3

1.1. Planteamiento del caso práctico inicial 3

/ 2. Definición del mapeo objeto relacional 4

/ 3. Ventajas e inconvenientes del mapeo objeto relacional 5

/ 4. Fases de mapeo objeto relacional 5

/ 5. Caso práctico 1: “Ventajas e inconvenientes de la adaptación a ORM” 6

/ 6. Herramientas ORM 7

/ 7. Definición de la arquitectura de Hibernate 8

/ 8. Componentes de Hibernate 9

/ 9. Instalación de Hibernate 9

/ 10. Configuración de Hibernate 10

/ 11. Caso práctico 2: “Log de Hibernate” 11

/ 12. Resumen y resolución del caso práctico de la unidad 11

/ 13. Webgrafía 12

OBJETIVOS

Aprender el concepto de mapeo objeto relacional

Conocer los principales ORM

Aprender a instalar Hibernate

/ 1. Introducción y contextualización práctica

En esta unidad, estudiaremos la capa más próxima a la información y a los datos desde nuestro aplicativo. Aprenderemos el concepto de mapeo objeto relacional y cómo simular entidades de base de datos en nuestro propio sistema.

Realizaremos una visión global del mercado descubriendo qué herramientas podemos encontrar que nos ayuden en esa simulación de entidades; es decir, nos aproximaremos a los principales frameworks que realicen este tipo de funcionalidades.

Por último, realizaremos una primera instalación del framework ORM que nos acompañará en el tema, y que nos ayudará a desarrollar ese concepto de mapeo objeto – relacional, denominado Hibernate. Para ello, seguiremos con el ejemplo en SpringBoot y aprenderemos a realizarlo en una aplicación web Java.

1.1. Planteamiento del caso práctico inicial

A continuación, vamos a plantear un caso práctico a través del cual podremos aproximarnos de forma práctica a la teoría de este tema.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema, encontrarás su resolución en el apartado Resumen y Resolución del caso práctico.

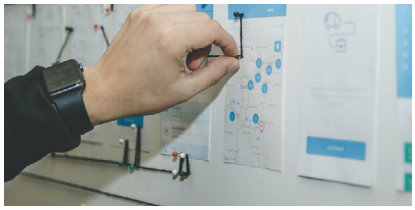

Fig. 1. Mapeo

Audio Intro. "Añadir Hibernate"

https://bit.ly/3fbWLQL

/ 2. Definición del mapeo objeto relacional

Como sabemos, **Java** es un lenguaje de programación orientado a objetos el cual puede ser representado en un **gráfico de objetos**; mientras que una **base de datos relacional** se representa en un formato tabular usando **filas y columnas.**

Cuando se trata de grabar un **objeto** en base de datos, existen algunas diferencias obvias entre estos dos sistemas; por lo tanto, para solventar este tipo de problemas o diferencias de datos a almacenar, tenemos la ayuda del **Mapeo Objeto Relacional.** Por lo tanto, podemos definir un ORM como un framework que facilita el almacenamiento de los objetos a una base de datos relacional.

Como bien sabemos, en cualquier base de datos, el nivel de elemento más alto es la **tabla**, dividida a su vez en **filas y columnas.** Una columna contiene valores en un tipo determinado, y una fila contendrá un conjunto de información de una tabla determinada.

Podríamos resumir exponiendo que un objeto plano es equivalente a una fila en una base de datos relacional, y básicamente esto es lo que se mapea en un sentido y en otro.

Audio 1. "ORM Hibernate Espejo"
https://bit.ly/2X8GyWi

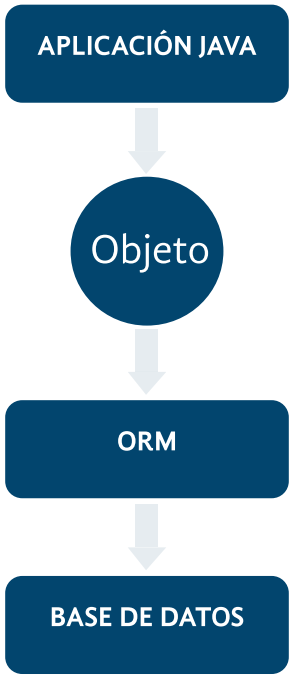

Fig. 2. Esquema Mapeo

En este gráfico podemos comprobar cuál sería el flujo del mapeo objeto relacional. Partimos de una aplicación con objetos desarrollados en Java, pasando por un framework que prepara esos objetos en entidades fáciles de persistir, y que se almacenan en base de datos.

/ 3. Ventajas e inconvenientes del mapeo objeto relacional

Tal y como estudiamos en unidades didácticas anteriores, Java posee una API de conexión a base de datos: los **drivers de conexión** (JDBC) para acceder a la base de datos. También vimos como nos facilita la forma de consultar dicha información en base de datos.

Si recordamos, escribiríamos ciertas consultas SQL nativas y obtendríamos un Resulset con el resultado de nuestra Query. Para hacer esto, el desarrollador debe conocer la base de datos a fondo, y saber qué tipo de relaciones existen entre tablas, así como el nombre exacto de las columnas, Constrains, etc.

Por otro lado, si consiguiéramos realizar las mismas operaciones con los datos desde la parte de Java, la perspectiva cambiaría totalmente. De esta forma tendríamos **abstracción, herencia, composición, identidad** y muchas características más en la parte de la aplicación Java que se podrían conseguir igualmente por medio de un framework (algunas de las razones por las que se usan los ORM).

Como sabemos, existen distintos tipos de base de datos, cada una con diferentes tipos de funciones y tipología de datos definidos. Cuando usamos la conexión JDBC debemos tener en cuenta este tipo de diferencias.

Algunas de las **ventajas** de usar un ORM son:

- Mejora en la **eficiencia** del desarrollo.
- **Desarrollo** más orientado a objetos.
- **Manejabilidad.**
- **Facilidad** para introducir nuevas funciones como el cacheo de información, por ejemplo.

Algunas de los **inconvenientes** que también hay que tener en cuenta son:

- **El mapeado automático** de las bases de datos consumen muchos recursos de sistema.
- **La sintaxis de los ORM** a veces puede complicarse si realizamos consultas muy complejas mediante las que crucemos varias tablas y con diversas condiciones.

Estudiaremos el funcionamiento del ORM en el siguiente apartado.

/ 4. Fases de mapeo objeto relacional

A continuación, haremos un breve resumen sobre la **arquitectura funcional de un framework ORM** contemplando tres fases de funcionamiento:

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

- **Fase 2:** esta fase es llamada también de persistencia o mapeo. Contiene los siguientes agentes:
  - **Proveedor JPA:** librería que hace posible toda la funcionalidad de JPA: javax.persistence.
  - **Archivo de asignación:** es un fichero XML donde se almacena la configuración de la asignación de los datos de una clase JAVA (POJO) y los datos reales de la base de datos relacional.
  - **JPA Cargador: r**ealmente esta parte funciona como una memoria caché, cargará los datos de la base de datos proporcionando algo parecido a una copia; para de esa forma, realizar interacciones rápidas con las clases de servicio.
  - **Reja de objeto:** es el lugar donde se almacenan temporalmente una copia de los datos de nuestra base de datos relacional. Se le llama objeto grid, por lo que todas las consultas pasarán por este punto, y una vez realizadas las verificaciones pasará a la base de datos principal.
- **Fase 3:** llamada fase de datos relacionales. Una vez pasada la reja de objetos y todo haya ido bien, se irá directamente a base de datos. Hasta entonces como hemos mencionado antes, se permanecerá en ese espacio temporal de caché.

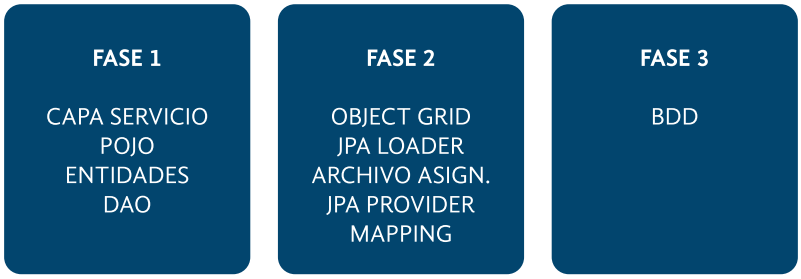

Fig. 3. Esquema arquitectura ORM

Vídeo 1. "Esquema ORM"
https://bit.ly/2EvTKy0

/ 5. Caso práctico 1: "Ventajas e inconvenientes de la adaptación a ORM"

Planteamiento: Disponemos de una aplicación web que actualmente está funcionando con conexión directa a base de datos relacional por medio del driver JDBC de conexión.

Las tablas con la que la aplicación está trabajando son: "clientes", "pedidos", "direcciones" y "empleados". Cada tabla contiene sus diferentes atributos y columnas.

Nudo: Se requiere realizar un análisis de las ventajas e inconvenientes de transformar el acceso directo a base de datos a un sistema con ORM.

Desenlace: Realizando una investigación sobre los distintos ORM del mercado se considera que Hibernate puede ser una buena solución para realizar el mapeo de dichos objetos.

Teniendo en cuenta que elegiremos un ORM para realizar accesos a datos, primero deberíamos realizar un análisis de las diferentes tablas y sus columnas, ya que se realizará una creación de estas, traducidas a entidades con sus atributos determinados en las diferentes tablas de clientes, pedidos, direcciones y empleados.

Así, algunas de las ventajas que conseguiríamos serían:

- Mejora del desarrollo orientado a objetos en esta parte.
- Mejora a la hora de realizar diferentes consultas y procedimientos.
- Mejora de la manejabilidad de los objetos del aplicativo.

Algunos de los inconvenientes podrían ser:

- Aumento de la memoria de la carga de recursos al introducir esta capa intermedia de Hibernate.
- Posibilidad de complicación de algunas consultas en formato de sintaxis del ORM, si son consultas muy complejas.

Fig. 4. Análisis de ventajas e inconvenientes

/ 6. Herramientas ORM

Dependiendo del lenguaje en el que estemos desarrollando vamos a encontrar diferentes frameworks recomendados (para .NET encontraremos unos, para PHP tendremos otros...). En este caso nos centraremos en los ORM más usados en Java:

- **EBEAN:** algunas de sus características son:
  - **Soporte en bases de datos:** soporta bases de datos como H2, Postgres, Mysql, NuoDB, PostGis, MariaDB, SQL Server, Oracle, SAP, etc.
  - **Múltiples niveles de abstracción:** consultas de tipo ORM mezcladas con SQL, además de consultas DTO.
  - **Beneficios:** evita automáticamente N+1, usa caché de tipo L2 para reducir carga de base de datos, realiza consultas mezclando base de datos y cacheado L2, ajusta automáticamente las consultas ORM, contiene tecnología Elasticsearch para caché L3, etc.
- **IBATIS:** un ORM que aparece de la mano de Apache Software Foundation. En 2010 el desarrollo se centralizó en "Google Code", usando el nuevo nombre MyBatis. Posee soporte para Java y .Net. Algunas de sus características son:
  - Posee la opción de dividir la capa de persistencia en: capa de abstracción, capa de framework persistente, y capa de Driver.
  - Una de sus virtudes en la facilidad de interactuar con los objetos y los datos de las bases de datos relacionales.
  - Ofrece abstracción a nivel de la capa de persistencia de objetos.
- **HIBERNATE:** es el ORM más extendido y más usado. Disponible para lenguaje Java y también para .Net (denominándose para éste, Nhibernate). Facilita el mapeo relacional de los distintos objetos entre una base de datos relacional y el modelo de objetos de la aplicación; para lo que se apoya en un fichero .xml que representa y establece dichas relaciones o también por medio de anotaciones donde se establecen las relaciones. Algunas de sus características:

- **Simplicidad:** al disponer de un solo fichero .xml para establecer las relaciones, es muy sencillo e intuitivo tanto dirigirnos a éste como consultar cualquier tipo de relación entre entidades o atributos.

- **Robusto:** dispone de muchas características adaptadas al lenguaje Java: colecciones, herencia, abstracción, orientación a objetos, etc. En la capa de abstracción ofrece una propia capa de consultas SQL llamada HQL, orientada a facilitar la sintaxis y a mejorar la eficiencia de estas.

/ 7. Definición de la arquitectura de Hibernate

A continuación, profundizaremos en Hibernate, un servicio de alta eficiencia de mapeo objeto-relacional y con funcionalidades bastante potentes de consulta. Empezaremos aprendiendo sobre su arquitectura y avanzaremos en la instalación y configuración de este Framework en aplicación Java.

La **arquitectura** de Hibernate incluye distintos objetos como son los **objetos de persistencia, sesiones, transacciones, entre otros.**

Hibernate usa el principio de **reflexión** Java, que básicamente permite el análisis y la modificación de los distintos atributos y características de las distintas clases en tiempo de ejecución.

A continuación, mostraremos un gráfico donde vemos una visión global de la arquitectura Hibernate y de las diferentes capas que lo conforman.

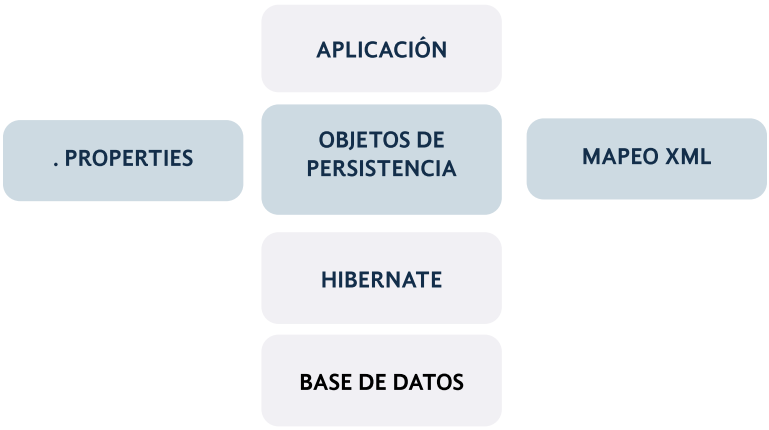

Fig. 5. Arquitectura Hibernate

Tal y como podemos ver, la capa de Aplicación e Hibernate están unidas por los Objetos de persistencia; esto es porque en una parte específica de la aplicación se da cierta conversión (fichero de mapeo), dónde la información fluye y es mapeada desde dichos ficheros persistentes a la base de datos.

Es en la capa Hibernate donde se realiza la conexión con el driver, también donde se cargan tanto las distintas configuraciones Hibernate, como todas las entidades previamente diseñadas.

/ 8. Componentes de Hibernate

Una vez vista la arquitectura de Hibernate a nivel muy básico, vamos a profundizar en algunas de las piezas clave que hacen posible el mapeo objeto relacional. Algunos de sus componentes principales son:

- **SessionFactory Object:** mediante el que se permitirá el uso de objetos de tipo Session. Este objecto Java se puede instanciar de diferentes formas: normalmente coge la configuración existente en el fichero de configuración establecido por defecto. Utilizaremos un objeto SessionFactory por cada base de datos que tengamos en la aplicación.

- **Session Object:** utilizaremos dicho objeto para instanciar una conexión directa con nuestra base de datos relacional. Es un objeto no muy pesado y su función principal es interactuar con la base de datos. Este tipo de objetos no deben permanecer abiertos mucho tiempo por temas de seguridad.

- **Transaction Object:** es un objeto opcional, que básicamente maneja las transacciones directamente con las bases de datos relacionales. Si no se quiere hacer uso de dicho objeto, también podemos indicar manualmente aquellos bloques que queremos que sean transaccionales. Recordemos de temas anteriores la definición de transacción orientada al bloque de operación/es, cuyo objetivo consistía en persistir todas y cada una de las operaciones que contenían dicho bloque, o por el contrario, realizar rollback (marcha atrás) en dicha operación.

- **Query Object:** en Hibernate disponemos de varias formas de realizar consultas a la base de datos. Este tipo de objetos utilizan consultas de tipo SQL o de tipo Hibernate Query (HQL). Con este tipo de objetos enlazaremos los distintos parámetros de nuestra consulta, podremos realizar ciertas restricciones como controlar el número de resultados, y ejecutar la consulta. Es un modo de realizar consultas mucho más dinámico que las consultas nativas.

- **Criteria Object:** desaparecerá el lenguaje nativo de SQL para dar paso a las consultas por medio de objetos Java, y por medio de las funciones que nos ofrece Hibernate, que más tarde serán traducidas a sentencias SQL.

/ 9. Instalación de Hibernate

Dependiendo del tipo de proyecto Java que tengamos construido, podremos instalar Hibernate de diferentes formas. Básicamente podremos añadir el .jar a nuestro proyecto con la versión específica de Hibernate que nos hayamos descargado. Si por el contrario estamos trabajando en un proyecto web con gestor de dependencias maven, podremos agregar nuestra dependencia de la versión determinada de Hibernate, y al compilar, se descargará dicha librería y tendremos las clases del framework disponibles para su uso.

En nuestro caso, y siguiendo los temas anteriores, realizaremos la instalación agregando la dependencia determinada usando **Spring Boot**, considerando que estamos realizando una aplicación desde cero, a la cual queremos agregar Hibernate. Si éste no fuese el caso, nos podríamos quedar simplemente con la parte de agregar la dependencia. Como ya hemos hecho anteriormente con las bases de datos embebidas, nos dirigiremos a la web inicializadora de arquetipos de Spring Boot. Hoy en día disponemos de la siguiente URL: https://start.spring.io/ Una vez hemos ingresado en la URL, nos centraremos solo en la parte de añadir el *framework* que vamos buscando, ya que la composición del proyecto SpringBoot la vimos en temas anteriores. Hacemos clic en ***Add dependencies*** (añadir dependencias).

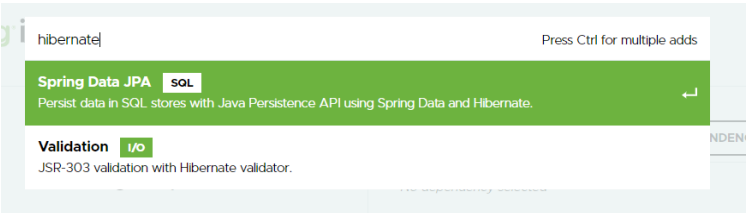

Fig. 6. Búsqueda de Hibernate

Como podemos ver en la imagen superior, buscando por Hibernate, añadiremos directamente JPA con Sping Data, e Hibernate y nuestras dependencias serían:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Fig. 7. Dependencia Hibernate

Aquí podemos observar la dependencia agregada. Si por el contrario quisiéramos agregar Hibernate sin tener a Spring Boot de mediador, podríamos dirigirnos a la web oficial de Hibernate y encontrar allí la última versión para la dependencia adecuada.

Vídeo 2. "Agregar Hibernate"
https://bit.ly/2CYhCKd

/ 10. Configuración de Hibernate

En este apartado, continuaremos con nuestra aplicación en Spring Boot y, por lo tanto, para realizar cambios de configuración nos dirigiremos al fichero "application.properties" de nuestro proyecto. A continuación, comentaremos algunos de los aspectos más importantes y usados en la configuración del framework instalado Hibernate.

```
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Fig. 8. Comunicación con BDD

Una vez situados en el fichero application.properties dispondremos de la línea que hemos colocado en la imagen superior, comunicando nuestro framework con la base de datos que tenemos instalada. En este caso es una base de datos H2, pero si tuviéramos una base de datos distinta, cambiaríamos y adaptaríamos la línea en función a la base de datos que tengamos.

Con la siguiente configuración, estaremos indicando que habilitamos las trazas de tipo SQL, y además, que se muestren con el formato SQL correspondiente.

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Fig. 9. Trazas SQL

Con las siguientes líneas de configuración, estaremos habilitando los logs de Hibernate a true; de esta forma podremos ver un log extenso sobre los errores que se vayan dando, así como las distintas líneas de log que queramos mostrar por nuestra cuenta. En la imagen también podemos observar que el nivel de log está establecido a debug (también podremos establecerlo a nivel info).

```
spring.jpa.properties.hibernate.generate_statistics=true
logging.level.org.hibernate.type=trace
logging.level.org.hibernate.stat=debug
```

Fig. 10. Logs

Con esta última línea, estaremos permitiendo a Hibernate crear manualmente distintas entidades que queramos generar en un fichero que habría que crear ("schema.sql"), y más tarde popular dichas tablas con otro script ("data.sql").

```
spring.jpa.hibernate.ddl-auto=none
```

Fig. 11. Inicialización BDD

/ 11. Caso práctico 2: "Log de Hibernate"

Planteamiento: estamos tratando con una aplicación web, la cual ya posee instalado el framework Hibernate. Desde el departamento de Dev-Ops (departamento de Desarrollo y Operaciones, Development y Operations) llega un ticket en el que nos piden que habilitemos los logs a nivel Info del framework; ya que dicho departamento, al realizar seguimiento de trazas de la aplicación, no puede ver las diferentes consultas ni tampoco las trazas de Logs que va dejando el propio Framework.

Investigado las distintas configuraciones, vemos que las trazas se hacen efectivas en determinados ficheros específicos de la aplicación.

Nudo: ¿Cómo podríamos darle respuesta al ticket?

Desenlace: para implementar la petición solicitada, se añadirán distintas líneas de comandos de configuración, orientadas para la visualización de trazas en Hibernate. Para ello, en primer lugar, se habilitarán los logs de Hibernate.

El siguiente requerimiento sería que se mostraran también las trazas de las distintas consultas y ordenes SQL que Hibernate está mandando a nuestra base de datos relacional, para ello configuraríamos:

```
spring.jpa.properties.hibernate.generate_statistics=true
logging.level.org.hibernate.type=trace
logging.level.org.hibernate.stat=info
```

Fig. 12. Loggin info

De esta forma habilitaremos trazar los diferentes logs de Hibernate, trazando así todos los logs de nivel info.

Todas las líneas de código que estén escritas para trazar "log.info" ahora aparecerán.

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Fig.13. Show sql

Finalmente, con estas dos líneas conseguiremos que las distintas consultas que realicemos desde nuestro framework Hibernate, sean mostradas también en el log genérico de la aplicación.

/ 12. Resumen y resolución del caso práctico de la unidad

En esta unidad didáctica, hemos aprendido el concepto de **mapeo objeto relacional,** hemos visto los diferentes elementos que componen el mapa de esta estructura y aprendimos qué es un **ORM** en nuestra aplicación.

También hemos realizado un recorrido entre los diferentes Frameworks ORM que podemos añadir a nuestro aplicativo Java, revisamos sus principales características, y nos quedamos con **Hibernate**, siendo éste el más popular, y el que más extiende su funcionalidad.

Por último, realizamos la **instalación de Hibernate** por medio de **Spring Boot** (usado en unidades didácticas anteriores), viendo también cómo podríamos realizar la instalación sin Spring Boot. También hemos aprendido las opciones más determinantes de configuración en esta herramienta y a cómo manipularlas.

Resolución del caso práctico de la unidad.

A nuestro compañero Roberto se le plantea una modificación en la aplicación web que desarrolla. Llega el momento de elegir un ORM, ya que ha caído en la cuenta de que a través de una herramienta ORM podrá gestionar mucho mejor la relación entre su aplicación y su base de datos relacional.

Para ello Roberto dispone de una solución simple:

Ya que Roberto no dispone del framework Spring-Boot, ni tampoco necesita montar desde cero el arquetipo de la nueva aplicación, la solución más práctica sería dirigirse al directorio oficial de dependencias de maven:

https://mvnrepository.com/

Una vez allí podrá realizar una búsqueda de la dependencia que desea agregar a su aplicación: en este caso buscaría por "Hibernate". Se podrá contemplar que existen numerosas versiones de Hibernate; por lo que lo recomendable es centrarse en la última released version que sea de tipo 'final', ya que este tipo de versiones han superado todos los controles a nivel de testing y ya se usan en el mercado:

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.4.18.Final</version>
</dependency>
```

Código 1. Dependencia Hibernate

Para este caso se ha elegido la versión que vemos arriba porque es la establecida como la última final. Con la versión ya disponible, Roberto solo tendría que aplicar los pasos de configuración.

/ 13. Webgrafía

https://mvnrepository.com/
https://hibernate.org/
https://www.oracle.com/es/java/
https://start.spring.io/