MEDAC
Instituto Oficial de Formación Profesional

ACCESO A DATOS

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Exploración del mapeo objeto-relacional

08

ÍNDICE

OBJETIVOS

Aprender qué es una clase persistente.

Conocer el fichero de mapeo.

Estudiar el objeto sesión.

Saber cómo obtener objetos, almacenarlos y modificarlos.

Aprender consultas y gestión de transacciones.

/ 1. Introducción y contextualización práctica

En esta nueva unidad, veremos el concepto de clase persistente y qué buenas prácticas podremos seguir para su preparación. También, mostraremos la estructura de un fichero XML básico de mapeo, viendo los distintos atributos y cómo realiza las relaciones entre objetos y base de datos.

Estudiaremos el objeto sesión, sus métodos principales y las sentencias más comunes vinculadas al mismo. Por último, veremos las distintas formas que existen para extraer datos, guardarlos e incluso modificarlos desde Hibernate.

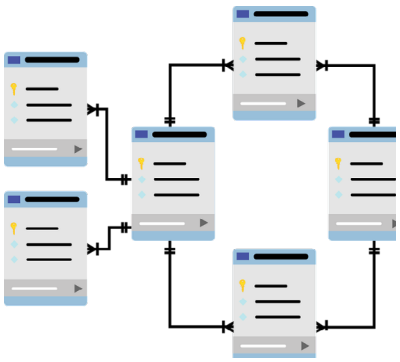

Fig1. Introducción

Planteamiento del caso práctico inicial

A continuación, vamos a plantear un caso 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.

Audio intro. "Mapeo de clase persistente"

https://bit.ly/3fnEr7q

/ 2. Clases persistentes

El concepto global de Hibernate, o un ORM en general, es coger ciertos valores desde los atributos de las clases de Java y persistirlos a las tablas de una base de datos. Un documento de mapeo nos ayudará a determinar cómo coger dichos valores desde las clases de Java y mapearlos a los campos asociados en base de datos. A las clases cuyos objetos serán almacenados en base de datos son llamadas **clases persistentes** en Hibernate. Este tendrá mejor eficiencia si estas clases de las que hablamos, siguen algunas reglas simples o el modelo de programación de objetos Java simples (POJO). A continuación, veremos algunas de las reglas principales de las clases persistentes, teniendo en cuenta que no son requerimientos puros:

- Todas las clases que van a ser persistidas necesitarán constructor por defecto.
- Todas las clases deben contener un atributo ID para facilitar la identificación de los objetos tanto en Hibernate como en base de datos. Será mapeado como primary key en base de datos.
- Todos los atributos de la clase deberán ser definidos como privados y tener métodos *get()* y *set()*.
- Una característica de las clases persistentes de *Hibernate* suele ser que dichas clases no sean de tipo "final".

Se usa el nombre de POJO para enfatizar que no es más que un objeto ordinario Java y no es ningún objeto especial.

```
public class Customer {
  private int id;
  private String firstName;
  private String lastName;
  private int customerNumber;
  public Customer() {}
  public Customer(String fname, String lname, int
custNum) {
    this.firstName = fname;
    this.lastName = lname;
    this.customerNumber = custNum;
  }
  public int getId() {
    return id;
  }
  public void setId( int id ) {
    this.id = id;
  }
  public String getFirstName() {
    return firstName;
  }
  public void setFirstName( String first_name ) {
    this.firstName = first_name;
  }
  public String getLastName() {
    return lastName;
  }
  public void setLastName( String last_name ) {
    this.lastName = last_name;
  }
  public int getCustomerNumber() {
    return customerNumber;
  }
  public void setCustomerNumber( int
customerNumber ) {
    this.customerNumber = customerNumber;
  }
}
```

Código 1. Ejemplo de clase persistente.

Vídeo 1. "Clases persistentes"

https://bit.ly/3gfwphW

/ 3. Fichero de mapeo I. Composición

Normalmente, las relaciones de mapeo objeto-relacional están definidas en un documento con extensión XML. Este documento guía a *Hibernate*, pero, claro, ¿cómo se mapean las clases previamente definidas a la base de datos? Aunque muchos desarrolladores que usan *Hibernate* prefieren escribir el fichero de mapeo a mano, realmente existen diversas herramientas que generan dicho documento. Entre otras, Xdoclet, *Middlegen* y, para desarrolladores algo más avanzados, *AndroMDA*. Partiendo de la clase Java POJO de la página anterior (*Código 1*), habría una tabla correspondiente a cada objeto que corresponde a la clase "Customer". En relación a dicho código que vimos anteriormente, para crear en base de datos dicha tabla, habría que ejecutar la siguiente sentencia:

```
create table CUSTOMER (
  id INT NOT NULL auto_increment,
  first_name VARCHAR(20) default NULL,
  last_name  VARCHAR(20) default NULL,
  customernsumber    INT  default NULL,
  PRIMARY KEY (id)
);
```

Código 2. DDL tabla Customer.

Teniendo en cuenta la clase Java que mostramos anteriormente, a continuación, veremos el fichero de mapeo, el cual le indicará a Hibernate cómo mapear dicha clase definida a la tabla de la base de datos:

```
<hibernate-mapping>
  <class name = "Customer" table = "CUSTOMER">

    <meta attribute = "class-description">
      This class contains the customer info.
    </meta>

    <id name = "id" type = "int" column = "id">
      <generator class="native"/>
    </id>

    <property name = "firstName" column = "first_name" type = "string"/>
    <property name = "lastName" column = "last_name" type = "string"/>
    <property name = "customerNumber" column = "customernumber" type = "int"/>

  </class>
</hibernate-mapping>
```

Código 3. Fichero de mapeo.

Este fichero lo guardaremos de la siguiente forma: "<nombre>.hbm.xml", por ejemplo *Customer.hmb.xml*.

Audio 1. "Anotaciones Hibernate"
https://bit.ly/39HMM4q

/ 4. Fichero de mapeo II. Análisis de elementos

Teniendo en cuenta el fichero de mapeo que se mostró en la página anterior, veamos los diferentes elementos de mapeo que se han usado:

- **El documento de mapeo** es un fichero XML que tiene como elemento principal ***<hibernate-mapping>***, el cual, en su interior, almacenará todas las clases definidas.

- **Los elementos de tipo *<class>*** son usados para definir mapeos especiales que van desde nuestras clases Java definidas a las tablas de base de datos. El nombre de la clase Java es especificado usando el atributo **name**, y la tabla asociada en base de datos es vinculada con el atributo **table.**

- **Los elementos *<meta>*** son opcionales y pueden ser usados, por ejemplo, para definir la descripción de una clase.

- **El elemento *<id>*** mapea el atributo ID único en la parte de la clase Java y lo transforma en Primary Key en la tabla de la base de datos. El atributo **name** hace referencia al atributo de la clase Java y **column** se refiere a la columna real que hay en la tabla de base de datos. El atributo **type** proporciona a Hibernate la tipología del objeto y será convertido desde tipología Java a SQL.

- **El elemento *<generator>***, junto con el **elemento id**, es usado para generar la clave primaria automáticamente. El atributo **class** del elemento *generator* se establece con el valor native para permitir a Hibernate crear la Primary Key con diferentes algoritmos: ***identity*, hilo o sequence**, dependiendo de las capacidades de la base de datos

- **El elemento *<property>*** se usa para mapear los atributos o propiedades de Java y transformarlos en columnas de la tabla asociada en la base de datos. El atributo **name** del elemento hace referencia al nombre del atributo en la clase Java y **column** se refiere a la columna que existe en base de datos. El atributo **type** proporciona y transforma la tipología del objeto Java a objeto de tipo SQL.

/ 5. Caso práctico 1: "Clase persistente"

Planteamiento: Se dispone de una clase "Vehículo" con los siguientes campos:

- Marca
- Motor
- Número de ruedas
- Kilómetros

Se requiere realizar una clase persistente con sus atributos, constructor por defecto y métodos *getter y setter.*

Nudo: Una vez estudiado el concepto de clase persistente, tendremos en cuenta las pautas estudiadas para definirla de tal forma que sea legible y sencilla.

Desenlace: Realizaremos una clase persistente teniendo en cuenta los cuatro campos que se requieren y uno más que será el id, numérico y autoincrementable para base de datos.

Definiremos los cinco atributos como privados, añadiremos dos constructores base: un constructor por defecto y otro con todos los atributos, y por último, añadiremos los métodos *get*() y *set*().

A continuación, vemos el código:

```
public class Vehiculo {
   private int id;
   private String marca;
   private String motor;
   private int numeroRuedas;
   private int numeroKilometros;

   public Vehiculo() {}
   public Vehiculo(String marca, String motor, int numeroRuedas, int numeroKilometros) {
      this.marca = marca;
      this.motor = motor;
      this.numeroRuedas = numeroRuedas;
      this.numeroKilometros = numeroKilometros;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getMarca() {
      return marca;
   }
   public void setMarca(String marca) {
      this.marca = marca;
   }
   public String getMotor() {
      return motor;
   }
   public void setMotor(String motor) {
      this.motor = motor;
   }
   public int getNumeroRuedas() {
      return numeroRuedas;
   }
   public void setNumeroRuedas(int numeroRuedas) {
      this.numeroRuedas = numeroRuedas;
   }
   public int getNumeroKilometros() {
      return numeroKilometros;
   }
   public void setNumeroKilometros(int numeroKilometros) {
      this.numeroKilometros = numeroKilometros;
   }
}
```

Código 4. Caso práctico clase persistente.

/ 6. Sesiones y objetos Hibernate I. Estados

Un objeto de sesión es usado para establecer una conexión física con una base de datos. Este objeto es ligero y diseñado para ser instanciado cada vez que se necesite una interacción con la base de datos. Los objetos persistentes son almacenados y devueltos a través del objeto de sesión **(Session object).**

El objeto de sesión **no debe mantenerse abierto durante mucho tiempo**, ya que podría ser peligroso por temas de seguridad y, por lo tanto, deben ser creados y destruidos cada vez que sea necesario utilizarlos.

La función principal de estos objetos es **ofrecer, crear, leer y borrar** operaciones para las instancias de las clases mapeadas. Dichas instancias pueden estar en uno de los siguientes estados:

- ***Transient*:** Nueva instancia de una clase persistente, no está aún asociada a un objeto de sesión y no tiene representación en la base de datos ni identificador según Hibernate.
- ***Persistent*:** Una instancia transient se puede hacer persistente asociándola con una sesión. Una instancia persistente tiene una representación en la base de datos, un identificador, y la asociación con el objeto de sesión.
- ***Detached*:** Una vez se cierra la sesión de Hibernate, la instancia persistente se convertirá en una instancia separada.

Una instancia de sesión es serializable si sus clases persistentes lo son. Una transacción puede tener el siguiente aspecto:

```
SessionFactory sessionFactory;
 Session session = sessionFactory.openSession();
 Transaction transaction = null;

try {
      transaction = session.beginTransaction();
      // Aquí realizaríamos operaciones
      transaction.commit();
   }

catch (Exception e) {
      if (transaction!=null) {
        transaction.rollback();
      e.printStackTrace();
        }
   } finally {
      session.close();
   }
```

Código 5. Ejemplo de uso de session Hibernate.

Si la sesión lanzara una excepción, se debería de hacer *roll-back* de la transacción y la sesión descartada.

/ 7. Sesiones y objetos Hibernate II. Métodos

Existen numerosos métodos en la interfaz Session, a continuación, veremos los más importantes y sus correspondientes funciones. Por supuesto en la página oficial de Hibernate https://hibernate.org/ podremos estudiar el resto:

- ***beginTransaction()*:** Básicamente, permite comenzar una unidad de trabajo y devolver el objeto asociado de la transacción.
- ***cancelQuery()*:** Cancela la ejecución de la consulta actual.
- ***Clear()*:** Elimina completamente la sesión.
- ***Close()*:** Finaliza la sesión liberando la conexión JDBC y limpiándola. Devuelve un objeto de tipo Connection.
- ***createCriteria(Class clasePersistente)*:** Crea una instancia nueva de Criteria para la clase persistente proporcionada como parámetro. Devuelve un objeto de tipo Criteria.
- ***createCriteria(String entityName)*:** Crea una instancia de tipo Criteria para la entidad que se le pasa como parámetro. Devuelve un objeto de tipo Criteria.
- ***getIdentifier(Object objeto)*:** Devuelve un objeto de tipo Serializable que es el identificador de la entidad proporcionada y asociada a esta sesión.
- ***createFilter(Object colección, String consulta)*:** Crea una nueva instancia de consulta en función a la colección pasada como parámetro y la consulta. Devuelve un objeto de tipo Query.
- ***createQuery(String consultaHQL)*:** Crea una instancia de consulta en función a la sentencia HQL que se le pasa como parámetro. Devuelve un objeto de tipo ***SQLQuery*.**
- ***delete(Object objeto)*:** Borra una instancia persistente del almacén de datos.
- ***getSessionFactory()*:** Devuelve un objeto de tipo SessionFactory, el cual creó la sesión actual.
- ***refresh(Object objeto):*** Vuelve a leer el estado de la instancia dada como objeto proveniente de la base de datos.
- ***isConnected()*:** Comprueba si la sesión está conectada actualmente.
- ***isOpen():*** Comprueba si la sesión aún está abierta.

/ 8. Carga, almacenamiento y modificaciones de objetos

En este apartado, veremos cómo podemos recuperar objetos de nuestra base de datos usando Hibernate, y también las diferentes formas para actualizarlos, guardarlos, etc. Para ello, disponemos de una serie de comandos que ejecutaremos desde nuestro objeto ***Session:***

- **Carga de objetos:** Como su nombre indica, con los siguientes comandos obtendremos datos de nuestra base de datos. Podremos usar:
  - ***Session.get():*** Devuelve un objeto persistente según los parámetros de objeto de entidad e identificador, y si no existe objeto en base de datos, devolverá null.
  - ***Session.load()*:** También devuelve un objeto persistente teniendo en cuenta los parámetros que se le pasan de entidad e identificador. Devolverá un ObjectNotFoundException en caso de no encontrar dicha entidad.

- **Almacenamiento de objetos:** Para guardar la información en base de datos desde Hibernate usaremos:
  - ***Session.persist()*:** Ejecuta el comando insert del lenguaje SQL almacenando filas en la base de datos. Este método es de tipo void(), no devuelve nada.
  - ***Session.save()*:** Igual que el método anterior, ejecuta internamente un método insert de SQL con la diferencia de que devuelve un objeto de tipo Serializable.
- **Modificación de objetos:** Con los siguientes comandos actualizaremos los diferentes objetos en base de datos.
  - ***Session.update()*:** Realizaremos un update en base de datos. Es el método primitivo para actualizar filas y necesita que haya otra instancia de session ejecutándose, y si no, lanzará una excepción.
  - ***Session.merge()*:** Se ejecutará un update también en base de datos, pero, en este caso, no tendremos que preocuparnos si existe ya una instancia ejecutándose, ya que este método realizará la operación gestionando el resto.
- **Otros métodos:**
  - ***Session.delete()*:** Pasaremos como parámetros la entidad persistente y se realizará el borrado en base de datos.
  - ***Session.saveOrUpdate()*:** Método de gran utilidad para permitir tanto la actualización de la entidad (si existe en base de datos) como el insert (si no existe en base de datos).

/ 9. Consultas SQL y HQL

El lenguaje *Hibernate Query* (HQL) es un lenguaje orientado a objetos muy parecido a SQL, pero en lugar de operar con tablas y columnas, HQL trabaja con objetos persistentes y con las propiedades de los mismos. Las consultas de tipo HQL son traducidas por Hibernate en consultas corrientes SQL, que más tarde son ejecutadas en base de datos.

En Hibernate, aunque podemos usar sentencias SQL directamente usando SQL nativas, se recomienda usar HQL para evitar problemas de portabilidad de la base de datos y aprovechar también las estrategias de caché implementadas en Hibernate.

A continuación, podemos ver un ejemplo de consulta HQL:

```
String hql = "FROM Customer";
Query consulta = session.createQuery(hql);
consulta.setFirstResult(1);
consulta.setMaxResults(40);
List results = consulta.list();
```

Código 6. Ejemplo HQL.

Este es un ejemplo muy sencillo de una consulta HQL donde consultamos la tabla "*Customer*", con la que obtenemos todos los registros, y más tarde, con "*setFirstResult*" y "*setMaxResults*", aplicamos un filtro de cuantas filas queramos obtener. En este caso, las 40 primeras. En la documentación oficial de Hibernate, podremos encontrar cómo realizar el resto de sentencias como *insert, delete, update*, etc.

También, podemos emplear sentencias de SQL nativas si queremos usar funcionalidades específicas de la base de datos que tenemos conectada a nuestra aplicación. Un ejemplo de ello puede ser la **ejecución de procedimientos almacenados en base de datos**, que son fragmentos de código o pequeñas aplicaciones desarrolladas en función al lenguaje de dicha base de datos. Se ejecutan dentro del motor de la base de datos relacional. Con estos procedimientos, nos ahorramos la recepción y el envío de datos al usuario, pudiendo suprimir la parte de la recepción realizando un pequeño informe a dicho usuario con las operaciones realizadas o con un reporte de resultados.

Continuando con el tema de las sentencias nativas SQL, podremos ejecutarlas desde nuestro código con el comando:

createSQLQuery()

Este método recibirá como parámetro una variable de tipo String que contendrá la consulta nativa SQL.

Devuelve un objeto de tipo ***SQLQuery*** y una vez obtenido dicho objeto, se puede asociar perfectamente a una entidad existente de Hibernate.

Profundizaremos más sobre consultas HQL en el siguiente vídeo y en el caso práctico 2.

Vídeo 2. "Consultas HQL"
https://bit.ly/2EDJvYv

/ 10. Gestión de transacciones con Hibernate

Una transacción representa una unidad de trabajo. De tal forma que si uno de los pasos falla, la transacción completa fallará (relacionado con el concepto de atomicidad). Una transacción, como hemos visto en temas anteriores, se define por las características ACID (Atomicidad, Consistencia, aislamiento (Isolation) y Durabilidad).

En Hibernate, tenemos la interfaz Transaction que define la unidad de trabajo. Mantiene la abstracción desde su propia implementación.

Algunos de los métodos principales en la interfaz *Transaction* son:

- ***Void begin*:** Empieza una transacción nueva.
- ***Void commit()*:** Finaliza la unidad de trabajo a menos que estemos en el modo "FlushMode.NEVER".
- ***Void rollback()*:** Fuerza a cancelar totalmente la transacción.
- ***Void setTimeout(int segundos)*:** Establece un time out determinado por parámetro en segundos.
- ***Boolean isAlive()*:** Comprueba si la transacción está aún activa.
- ***Void registerSynchronization(Synchronization s)*:** Registra la respuesta sincronizada de un usuario para esa transacción.
- ***Boolean wasCommited()*:** Comprueba si se ha cerrado la transacción satisfactoriamente.
- ***Boolean wasRolledBack()*:** Comprueba si la transacción ha sido cancelada satisfactoriamente.

Fig. 2. Pasos de una transacción.

/ 11. Caso práctico 2: "Sentencias HQL"

Planteamiento: Disponemos de la clase anteriormente descrita en el tema, denominada "*Customer*", cuyos campos son:

```
 private int id;
private String firstName;
private String lastName;
private int customerNumber;
```

Código 7. Campos

Se requiere realizar una sentencia HQL de tipo *update* en la entidad *Customer* y en la columna, "*customerNumber*", para establecer en este campo el número 25 como número de cliente, siempre y cuando su "Id" sea 105 y cuando su "firstName" sea "Pepe".

Nudo: Usaremos el potencial de Hibernate para realizar una sentencia HQL por medio de parámetros.

Desenlace: A continuación, veremos cómo se quedaría el código y lo explicaremos más tarde. En el vídeo 2, también puedes encontrar parte de la resolución.

```
 Session session = sessionFactory.openSession();
   Transaction transaction = session.beginTransaction();
Query q=session.createQuery(
     "update Customer set CUSTOMERNUMBER=:customerNumber where ID=:id and FIRSTNAME=:firstName");
q.setParameter("customerNumber",25 );
q.setParameter("id",105 );
q.setParameter("firstName", "Pepe");

int status=q.executeUpdate();
transaction.commit();
if (status > 0)
   System.out.println("Update realizado");
else
   System.out.println("Update no realizado");
```

Código 8. Caso práctico 2. Sentencias HQL.

Aquí podemos ver cómo instanciamos nuestro objeto *session*, también cómo iniciamos una transacción y cómo, con el propio objeto *session* y su método createQuery, asignamos a una variable de tipo Query la consulta HQL que vamos a preparar.

Podemos observar que vamos a persistir sobre la tabla "Customer", y que disponemos de tres parámetros: con el primero 'seteamos' el valor a la columna "CUSTOMERNUMBER", el segundo y el tercero entran dentro de la cláusula WHERE, donde filtramos por id para que se haga coincidir con el 105, y "firstName", para que sea "Pepe". De esta forma, podemos ver abajo los parámetros definidos.

Se ejecuta la query y guardamos el valor resultante en una variable de tipo entero, si es 1, quiere decir que ha sido persistida una tabla, y, por tanto, es correcto.

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

En este tema, hemos aprendido el concepto de **clase persistente.** Hemos visto sus principales características y mostramos un ejemplo que nos ayudará a guiarnos para crear clases propias.

Hemos comprobado la importancia del **fichero de mapeo**, concretamente del enlace que realiza desde los objetos Java, y cómo los transforma en información efectiva almacenada en base de datos. También, estudiamos el **objeto *session*** y los métodos más importantes que lo rodean.

Por último, hemos comprobado las diferentes formas que existen de ejecutar sentencias desde Hibernate: SQL y HQL, y cómo gestionar **transacciones** desde nuestro framework.

Resolución del caso práctico de la unidad.

A nuestro compañero Carlos se le ha entregado una tarea para realizar el mapeo de la clase PERSONAS. Dicha clase contiene cuatro atributos: **Id, nombre, edad y altura.**

Para ello, Carlos deberá ser conocedor tanto de los atributos de la clase persistente como de los nombres reales de las columnas en base de datos, así como los tipos definidos en ella.

```
<hibernate-mapping>
  <class name = “Personas” table = “PERSONAS”>

    <meta attribute = “class-description”>
      Esta clase contiene características de un humano.
    </meta>

    <id name = “id” type = “int” column = “id”>
      <generator class=”native”/>
    </id>

    <property name = “nombre” column = “name” type = “string”/>
    <property name = “edad” column = “age” type = “ int “/>
    <property name = “altura” column = “height” type = “int”/>

  </class>
</hibernate-mapping>
```

Código 9. Mapeo de la clase Personas.

Con este fragmento de código sería suficiente para mapear la entidad Personas a la tabla PERSONAS de la base de datos.

Algunos puntos a señalar son:

- Dentro de la etiqueta <meta>, hemos usado el atributo “class description” para escribir una breve descripción de la entidad.

- Hemos tenido especial cuidado de colocar bien escritos tanto los atributos de la clase persistente como las columnas de base de datos y sus tipos.

/ 13. Webgrafía

https://hibernate.org/
https://www.oracle.com/es/java/