IES CHAN DO MONTE
C.S. de Desarrollo de Aplicaciones Multiplataforma

ANEXO:

UNIDAD 4: HIBERNATE –HERRAMIENTA ORM-

Índice

1. ACLARACIÓN DE LA OPCIÓN CASCADE

Se quiere crear un método en Hibernate que reciba un objeto Departamento nuevo y una colección de Proxecto y los inserte en la base de datos.

SIN LA OPCION DE CASCADE AL MAPEARLO

Aquí tienes un ejemplo de cómo podría ser:

En este ejemplo: En la colección set no está definida la propiedad cascade

```java
@Entity
@Table(name = "Proxecto")
public class Proxecto implements Serializable {

    @Id
    @Column(name = "NumProxecto")
    private int numProxecto;

    @Column(name = "NomeProxecto")
    private String nomeProxecto;

    @Column(name = "Lugar")
    private String lugar;

    @ManyToOne
    @JoinColumn(name = "NumDepartControla")
    private Departamento departamento;

    // getters and setters
}
```

```java
@Entity
@Table(name = "Departamento")
public class Departamento implements Serializable {

    @Id
    @Column(name = "NumDepartamento")
    private int numDepartamento;

    @Column(name = "NomeDepartamento")
    private String nomeDepartamento;

    @Column(name = "Director")
    private String director;

    @OneToMany(mappedBy = "departamento")
    private Set<Proxecto> proxectos;

    // getters and setters
}
```

Ficheros de mapeo equivalentes

Fichero de mapeo para `Proxecto` (`Proxecto.hbm.xml`):

XML

```xml
<hibernate-mapping>
    <class name="Proxecto" table="Proxecto">
        <id name="numProxecto" column="NumProxecto" type="int">
            <generator class="assigned"/>
        </id>
        <property name="nomeProxecto" column="NomeProxecto" type="string"/>
        <property name="lugar" column="Lugar" type="string"/>
        <many-to-one name="departamento" column="NumDepartControla" class="Departamento"
fetch="join"/>
    </class>
</hibernate-mapping>
```

Fichero de mapeo para `Departamento` (`Departamento.hbm.xml`):

XML

```xml
<hibernate-mapping>
    <class name="Departamento" table="Departamento">
        <id name="numDepartamento" column="NumDepartamento" type="int">
            <generator class="assigned"/>
        </id>
        <property name="nomeDepartamento" column="NomeDepartamento" type="string"/>
        <property name="director" column="Director" type="string"/>
        <set name="proxectos" inverse="true">
            <key column="NumDepartControla"/>
            <one-to-many class="Proxecto"/>
        </set>
    </class>
</hibernate-mapping>
```

En Hibernate, la propiedad **inverse** se utiliza para definir el "**lado propietario" de una relación**.

El lado propietario es responsable de actualizar la relación en la base de datos.

Si **inverse** se establece en **true**, significa que **la entidad en la que se encuentra esta propiedad NO es el lado propietario de la relación**. En tu caso, al establecer inverse="true" en la entidad Departamento, estás diciendo que Proxecto es el lado propietario de la relación, y por lo tanto, es responsable de actualizar la relación en la base de datos.

En las **anotaciones de JPA/Hibernate**, el concepto de inverse se maneja de manera un poco diferente. En lugar de tener una propiedad inverse, el "lado propietario" se determina por el lugar donde se coloca la anotación **@JoinColumn**.

La entidad que tiene la anotación @JoinColumn es el lado propietario de la relación.

Al colocar @JoinColumn(name = "NumDepartControla") en la entidad Proxecto, estás haciendo que Proxecto sea el lado propietario de la relación, que es el mismo comportamiento que inverse="true" en los ficheros de mapeo XML.

Por lo tanto, no se necesita una propiedad inverse en las anotaciones porque el lado propietario se infiere de la ubicación de la anotación @JoinColumn.

Es importante elegir correctamente el lado propietario de una relación, ya que puede afectar el rendimiento de tu aplicación y la coherencia de los datos. En general, el lado propietario debería ser el lado que tiene la cardinalidad "muchos" en una relación uno-a-muchos (Proxecto sería el lado propietario ideal de la relación con Departamento).

Si no se tiene la opción de cascada en persist o save, se tendrá que guardar manualmente cada objeto Proxecto en la base de datos. Aquí tienes un ejemplo de cómo se haría:

```
public void insertarDepartamentoYProxectos(Session session, Departamento departamento,
Set<Proxecto> proxectos) {
    Transaction tx = null;
    try {
        tx = session.beginTransaction();

        // Guarda el departamento en la base de datos
        session.save(departamento);

        // Asigna los proyectos al departamento y los guarda uno por uno
        for (Proxecto proxecto : proxectos) {
            proxecto.setDepartamento(departamento);
            session.save(proyecto);
        }

        tx.commit();  // Confirma la transacción
    } catch (Exception e) {
        if (tx != null) tx.rollback();  // Si algo sale mal, deshace los cambios
        throw e;
    } finally {
        session.close();  // Cierra la sesión
```

La línea **proxecto.setDepartamento(departamento);** es necesaria para establecer la relación bidireccional entre Proxecto y Departamento.

Un Proxecto **tiene una referencia a un Departamento** (indicado por el campo departamento en la clase Proxecto). Cuando se guarda un Proxecto, este Proxecto está asociado con un Departamento específico en la base de datos. Para hacer esto, se debe decirle al Proxecto cuál es elDepartamento está asociado con él. Esto se hace con proxecto.setDepartamento(departamento);.

Por otro lado, un Departamento puede tener muchos Proxecto. Cuando se guarda el Departamento, se le dice que Proxectos están asociados con él (**departamento.setProxectos(proxectos)**;). Pero cada Proxecto en esa lista todavía necesita saber que está asociado con este Departamento.

En resumen, proxecto.setDepartamento(departamento); es necesario para mantener la consistencia y asegurar que la relación bidireccional entre Proxecto y Departamento se mantiene correctamente.

CON LA OPCION DE CASCADE AL MAPEARLO

```
@Entity
@Table(name = "Departamento")
public class Departamento implements Serializable {

    @Id
    @Column(name = "NumDepartamento")
    private int numDepartamento;

    @Column(name = "NomeDepartamento")
    private String nomeDepartamento;

    @Column(name = "Director")
    private String director;

    @OneToMany(mappedBy = "departamento", cascade = {CascadeType.PERSIST, CascadeType.MERGE,
CascadeType.SAVE_UPDATE})
    private Set<Proxecto> proxectos;

    // getters and setters
}
```

Fichero de mapeo Departamento.hbm.xml equivalente.

```
<hibernate-mapping>
    <class name="Departamento" table="Departamento">
        <id name="numDepartamento" column="NumDepartamento" type="int">
            <generator class="assigned"/>
        </id>
        <property name="nomeDepartamento" column="NomeDepartamento" type="string"/>
        <property name="director" column="Director" type="string"/>
        <set name="proxectos" cascade="save-update,merge,persist">
            <key column="NumDepartControla"/>
            <one-to-many class="Proxecto"/>
        </set>
    </class>
</hibernate-mapping>
```

Este código define una entidad Departamento con una relación OneToMany con Proxecto. Las operaciones de guardar, actualizar y fusionar se propagarán desde Departamento a Proxecto, pero la eliminación de un Departamento no eliminará los Proxecto asociados

```
public void insertarDepartamentoYProxectos(Session session, Departamento departamento,
Set<Proxecto> proxectos) {
    Transaction tx = null;
    try {
        tx = session.beginTransaction();

        // Asigna los proyectos al departamento
        departamento.setProxectos(proxectos);

        // Guarda el departamento en la base de datos
        // Esto también guardará los proyectos gracias a la opción CascadeType.PERSIST
        session.persist(departamento);

        tx.commit();  // Confirma la transacción
    } catch (Exception e) {
        if (tx != null) tx.rollback();  // Si algo sale mal, deshace los cambios
        throw e;
    } finally {
        session.close();  // Cierra la sesión
    }
}
```

Cuando se configuras las **opciones de cascada en la relación @OneToMany en Departamento**, Hibernate se encarga automáticamente de establecer la asociación entre Proxecto y Departamento cuando se guarda el Departamento. Esto significa que **Hibernate invoca automáticamente proxecto.setDepartamento(departamento) para cada Proxecto** en la colección que estás guardando.

Esto es muy útil porque te ahorra tener que hacerlo manualmente y ayuda a mantener la consistencia en tu modelo de datos. Sin embargo, es importante recordar que esto solo ocurre si has configurado las opciones de cascada adecuadas. Si no has configurado las opciones de cascada, tendrías que establecer la asociación manualmente.

2. PERSIST VERSUS SAVEORUPDATE

Hibernate es una implementación de la especificación de Java Persistence API (JPA), por lo que conserva los métodos **persist y merge de JPA**. Además, Hibernate **añade sus propios métodos**, como **save y saveOrUpdate**.

- **persist**: Este método de JPA almacena un objeto transient en el contexto de persistencia y al hacer commit se inserta este objeto nuevo en la base de datos. El objeto debe estar en estado "transient" (es decir, no asociado a ninguna sesión (no está en el contexto de persistencia y no representando ninguna fila en la base de datos).
- **merge**: Este método de JPA actualiza un objeto existente en la base de datos. Se utiliza principalmente con objetos en estado "detached" (es decir, que fueron previamente asociados a una sesión, pero esa sesión ha sido cerrada).
- **save**: Este método de Hibernate también JPA almacena un objeto transient en el contexto de persistencia y al hacer commit se inserta este objeto nuevo en la base de datos similar a persist. Sin embargo, save devuelve el identificador generado para el objeto.
- **saveOrUpdate**: Este método de Hibernate es flexible: si el objeto es nuevo, lo guarda en la base de datos; si el objeto ya existe en la base de datos, lo actualiza.

Si se intenta llamar a **persist en un objeto que ya está en el contexto de persistencia** (es decir, ya está asociado a la sesión actual), **Hibernate lanzará una excepción**. Esto se debe a que persist está diseñado para ser utilizado con objetos nuevos que aún no están en la base de datos ni en el contexto de persistencia.

Por otro lado, saveOrUpdate es más flexible. **Si el objeto ya está en el contexto de persistencia, saveOrUpdate simplemente actualizará cualquier cambio que hayas hecho en el objeto cuando se haga "commit" de la transacción**.

Si el objeto no está en el contexto de persistencia, pero existe en la base de datos, saveOrUpdate también actualizará su estado en la base de datos. Y si el objeto es nuevo y no existe en la base de datos, saveOrUpdate lo guardará en la base de datos al hacer el commit.