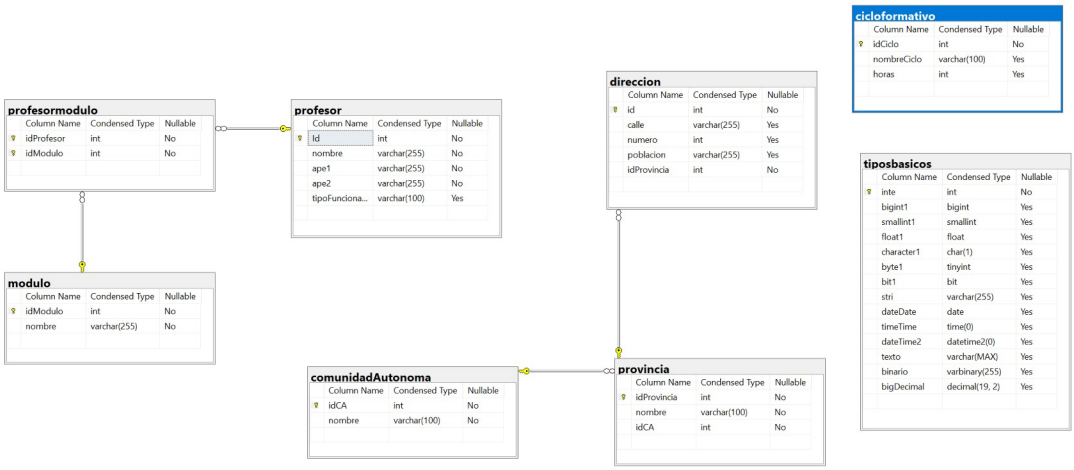
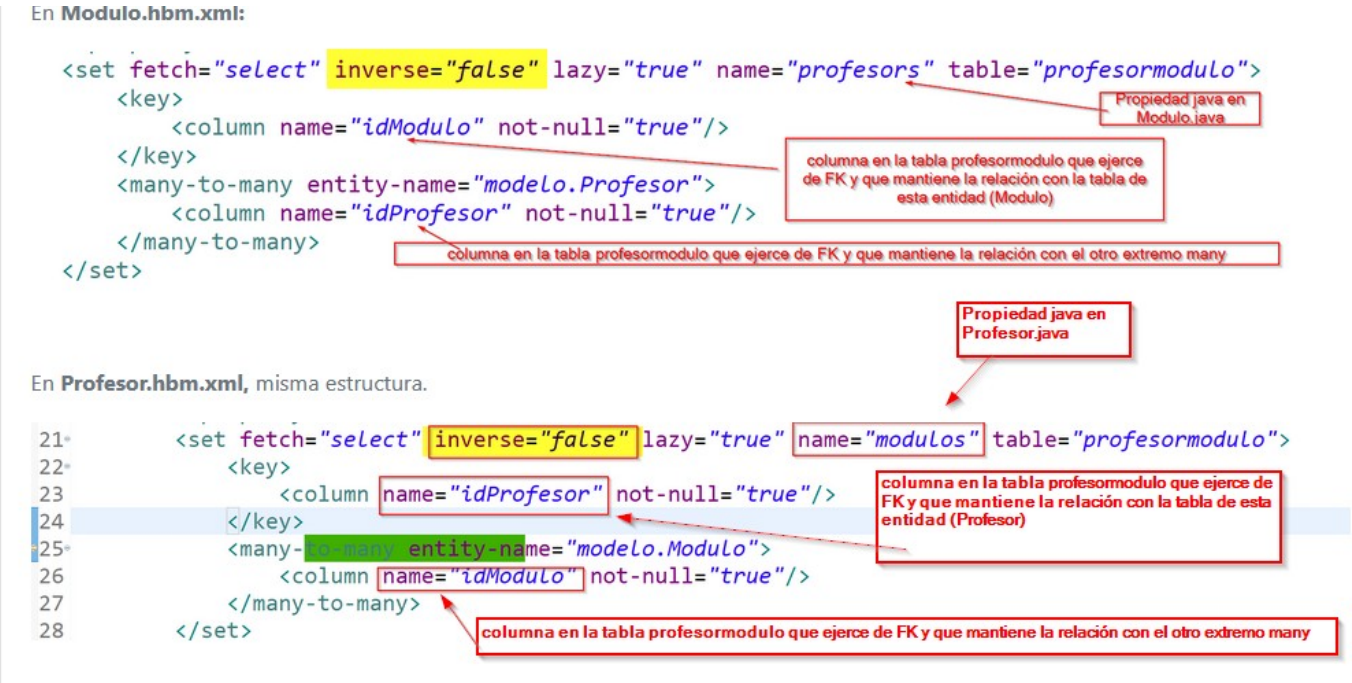
Mapeo de relaciones M:N

Una vez ejecutados los scripts 4 y 5 de la BD instituto de la carpeta https://www.edu.xunta.gal/fpadistancia/mod/folder/view.php?id=517029 , tendremos nuevas tablas y relaciones:



En la BD de instituto ampliada, existe una relación M:N entre *Profesor y Modulo.*
Si siguiésemos los pasos de la Actividad 3.1 desde el paso 13 en adelante, sobre la nueva BD de instituto, el plugin de Eclipse generaría el proyecto disponible en GitHub https://github.com/add-code-2223/UD3_EjemplosJoinsHQL[1] lo siguiente:

1 Sobre las clases generadas del plugin de Hibernate para Eclipse se ha hecho la modificación de renombrar la clase Cicloformativo.java a Ciclo.java y se ha cambiado el atributo nombreCiclo a nombre.



Fijémonos en que el plugin no atribuye el lado propietario a ningún extremo. (ambos elementos <set> tienen un atributo **inverse="false"**).

Revisitiemos de nuevo Relaciones en archivos de mapeado *.hbm.xml (página 4 y 5) de UD3 Herramientas de mapeamento objeto-relacional (ORM) V3.pdf

¡ATENCIÓN! La nomenclatura en XML de Hibernate puede llevar a confusión: con el atributo inverse=true en una colección se indica que la entidad que representa esa colección es el lado propietario.

Probemos a intentar crear una relación entre un profesor y un módulo y observemos el resultado:

Actividad 3.5 -> Ver método **crearModuloProfeNuevos**

Tal y como está, no se guardará salvo que un lado sea lado propietario.

Concepto Proxy y Lazy initialization

Para evitar que cuando un objeto se recupera de BD, se recuperen también todos los relacionados con éste, la documentación de Hibernate aconseja, en general, el uso de FetchType.LAZY en todo tipo de relaciones.
De esta manera, por ejemplo, se evita que cuando se recupere un departamento, se recuperen también su jefe y sus empleados

En el caso de relaciones con carga lazy, en lugar del objeto relacionado se carga un objeto conocido como **proxy.** Los proxies son generados por Hibernate en tiempo de ejecución y son clases que extienden a la clase original.

Por ejemplo, cuando se trae un departamento de BD, se trae un *proxy* que hereda de Emp para su jefe y una colección especial de Hibernate para sus empleados.

Cuando se invocan operaciones sobre un *proxy* no inicializado o una colección no inicializada, Hibernate inicializa el proxy o la colección.

Un proxy **siempre tiene el atributo clave correctamente inicializado** => department.getEmp().getEmpno() no causa que se cargue el objeto Emp del jefe del departamento.

Sin embargo, cuando se ejecuta department.getEmp().getNombre() Hibernate inicializa el proxy del empleado director de ese departamento => lanza una consulta para recuperar los datos de ese empleado director

Establecimiento de relaciones bidireccionales en relaciones M:N

Hibernate no gestiona automáticamente el establecimiento de relaciones bidireccionales

Para evitar problemas es buena práctica establecer las relaciones bidireccionales explícitamente de manera sistemática:

En **Profesor.java** crearemos un método:

```
public void addModulo(Modulo mod) {
    getModulos().add(mod);
    //conviene usar getModulos() en lugar de this.modulos para forzar las inicializaciones de los objetos proxy en caso de inicialización //lazy
    mod.getProfesors().add(this);
}
```

En **Modulo.java** crearemos su contrapartida:

```
public void addProfesor(Profesor profe) {
    getProfesors().add(profe);
```

```
        //conviene usar getProfesors() en lugar de this.profesors para forzar las inicializaciones de los objetos proxy en caso de inicialización //lazy
        profe.getModulos().add(this);
    }
```

Para añadir una relación bidireccional, se llamará a uno de los dos métodos que acabamos de ver y, a continuación, a los save de las 2 entidades.

Para añadir la relación, **basta con llamar a uno de los dos métodos**:

```
addProfesor(Profesor profe);
addModulo(Modulo mod);
```

En Actividad 3.5**, se corresponde con el apartado 1)e).**

Establecimiento de relaciones bidireccionales en relaciones 1:N

En relaciones 1:N, es más razonable añadir un método en la parte 1 que gestione la relación bidireccional, por ejemplo, entre ComunidadAutonoma y provincia:

En **ComunidadAutonoma.java**:

```
    public void addProvincia(Provincia prov) {
        getProvincias().add(prov); //uso de getProvincias() en lugar de this.provincias para forzar inicialización del proxy
        prov.setComunidadAutonoma(this);
    }
```

Operaciones en cascada o persistencia transitiva

En las relaciones es posible establecer qué operaciones se desean aplicar también a los

entidades relacionadas con una entidad en particular.

Es especialmente útil en relaciones 1:N o 1:1. (En M:N no se aconseja por si existen otras entidades relacionadas en los otros extremos que pueden complicar la transitividad.)

Se añade en el fichero de mapeado de la entidad que representa el extremo 1 que debe conllevar las operaciones de borrado, creación, etc. en la entidad del extremo opuesto. Por ejemplo, cuando queramos eliminar una comunidad autónoma, sería lógico eliminar también sus provincias.

En **ComunidadAutonoma.hbm.xml,** se añade el atributo **cascade** a la etiqueta que representa la relación con el otro extremo :

```
12        <set fetch="select" inverse="true" lazy="true" name="provincias" table="provincia"
13        cascade="all">
14            <key>
15                <column name="idCA" not-null="true"/>
16            </key>
17            <one-to-many class="modelo.Provincia"/>
18        </set>
```

El valor del atributo cascade dependerán de si se está usando JPA o los tipos específicos de Hibernate. Nosotros usaremos los tipos específicos de Hibernate:

`save-update, delete, all, delete-orphan`

- `save-update`: Si se crea/actualiza una entidad, se crean/actualizan sus asociadas
- `delete`: Si se elimina una entidad, se eliminan sus asociadas
- `all`: Se aplicarán todas las operaciones a todas las asociaciones, salvo delete-orphan
- `delete-orphan:` Se aplica solo a asociaciones de 1:N e indica que la operación de borrado debe aplicarse a cualquier objeto relacionado que se elimine de la asociación. all no la incluye (Tendría que usarse cascade="all,delete-orphan")

Se podrían combinar uno o varios valores separados por comas:
cascade="save-update,delete"
Esto se puede observar en los subapartados de la sección 2) de la Actividad3.8.

Más información: https://docs.jboss.org/hibernate/core/3.3/reference/en/html/objectstate.html#objectstate-transitive