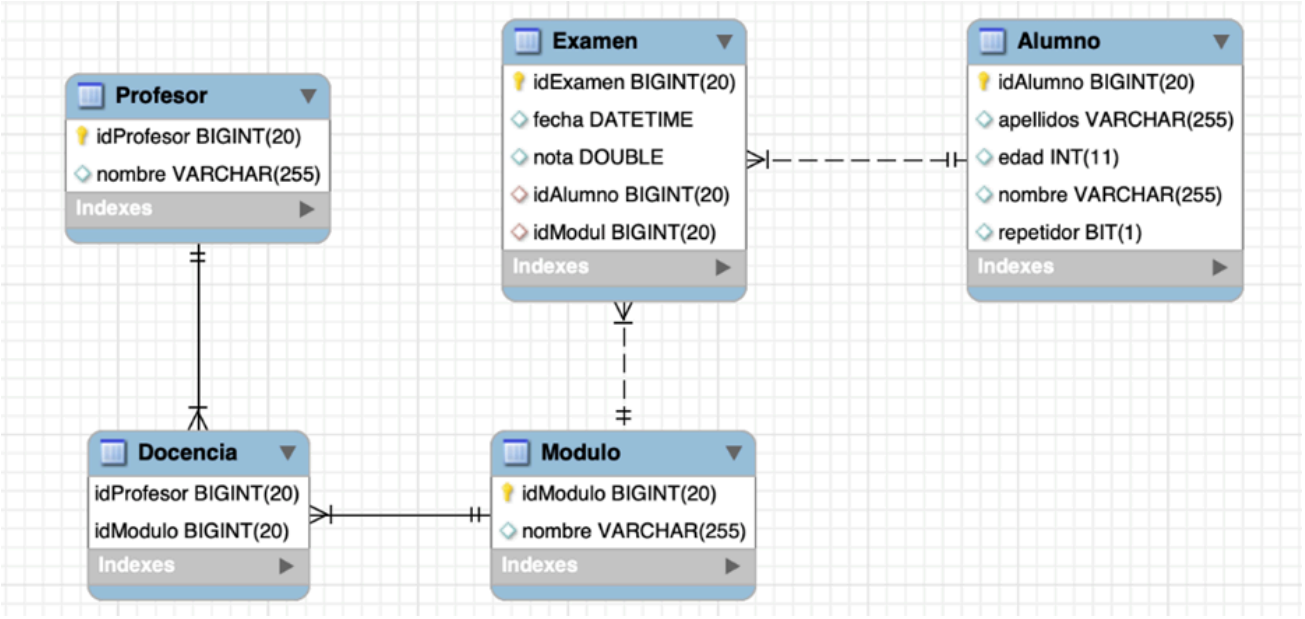
UT3. MAPEO OBJETO-RELACIONAL
Proyecto Ejemplo 5: Consultas HQL SOBRE BD ALUMNO-EXAMEN-MODULO-PROFESOR

1. Crear base de datos y tablas en MySQL Workbench

Vamos a ver algunos ejemplos de consultas HQL sobre una base de datos nueva, que llamaremos ut3demo5. El esquema es el siguiente:



Ejecutamos el siguiente script SQL en MySQL Workbench para crear la base de datos y todas las tablas

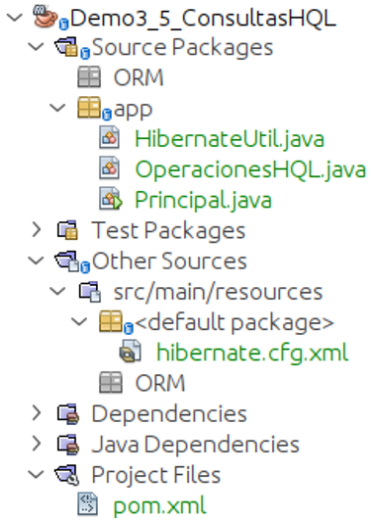
```
DROP DATABASE IF EXISTS ut3demo5;
CREATE DATABASE ut3demo5;
USE ut3demo5;
--
-- Table structure for table `Profesor`
--
CREATE TABLE `Profesor` (
  `idProfesor` bigint(20) NOT NULL AUTO_INCREMENT,
  `nombre` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`idProfesor`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4;
--
-- Dumping data for table `Profesor`
--
INSERT INTO `Profesor` VALUES
(1,'Pedro Faus'),
(2,'Ana Marto'),
(3,'Saturnino Perez'),
(4,'Ángel Sanz');
--
-- Table structure for table `Modulo`
--
```

```
CREATE TABLE `Modulo` (
  `idModulo` bigint(20) NOT NULL AUTO_INCREMENT,
  `nombre` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`idModulo`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4;
--
-- Dumping data for table `Modulo`
--
INSERT INTO `Modulo` VALUES (1,'PRG'),(2,'FOL'),(3,'EIE'),(4,'PMDM'),(5,'BBDD'),(6,'AD');
--
-- Table structure for table `Docencia`
--
CREATE TABLE `Docencia` (
  `idProfesor` bigint(20) NOT NULL,
  `idModulo` bigint(20) NOT NULL,
  PRIMARY KEY (`idProfesor`,`idModulo`),
  KEY `FK_DOC_MOD` (`idModulo`),
  CONSTRAINT `FK_DOC_MOD` FOREIGN KEY (`idModulo`) REFERENCES `Modulo` (`idModulo`),
  CONSTRAINT `FK_DOC_PROF` FOREIGN KEY (`idProfesor`) REFERENCES `Profesor` (`idProfesor`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
--
-- Dumping data for table `Docencia`
--
INSERT INTO `Docencia` VALUES (2,1),(1,2),(1,3),(1,4),(2,4),(3,6),(4,6);
--
-- Table structure for table `Alumno`
--
CREATE TABLE `Alumno` (
  `idAlumno` bigint(20) NOT NULL AUTO_INCREMENT,
  `apellidos` varchar(255) DEFAULT NULL,
  `edad` int(11) DEFAULT NULL,
  `nombre` varchar(255) DEFAULT NULL,
  `repetidor` bit(1) DEFAULT NULL,
  PRIMARY KEY (`idAlumno`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4;
--
-- Dumping data for table `Alumno`
--
INSERT INTO `Alumno` VALUES
(1,'Gómez Blas',24,'Pedro',1),
(2,'Pons Pont',20,'Pere',0),
(3,'Bo Pla',21,'Ana',0),
(4,'Martínez Garcia',24,'Pedro',0),
(5,'Gimeno Pons',19,'Saul',1),
(6,'Zaragoza Vila',24,'Alexia',0);
--
-- Table structure for table `Examen`
--
CREATE TABLE `Examen` (
  `idExamen` bigint(20) NOT NULL AUTO_INCREMENT,
  `fecha` datetime DEFAULT NULL,
  `nota` double DEFAULT NULL,
  `idAlumno` bigint(20) DEFAULT NULL,
  `idModul` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`idExamen`),
  KEY `FK_EX_ALU` (`idAlumno`),
  KEY `FK_EX_MOD` (`idModul`),
  CONSTRAINT `FK_EX_ALU` FOREIGN KEY (`idAlumno`) REFERENCES `Alumno` (`idAlumno`),
  CONSTRAINT `FK_EX_MOD` FOREIGN KEY (`idModul`) REFERENCES `Modulo` (`idModulo`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4;
--
-- Dumping data for table `Examen`
--
INSERT INTO `Examen` VALUES
```

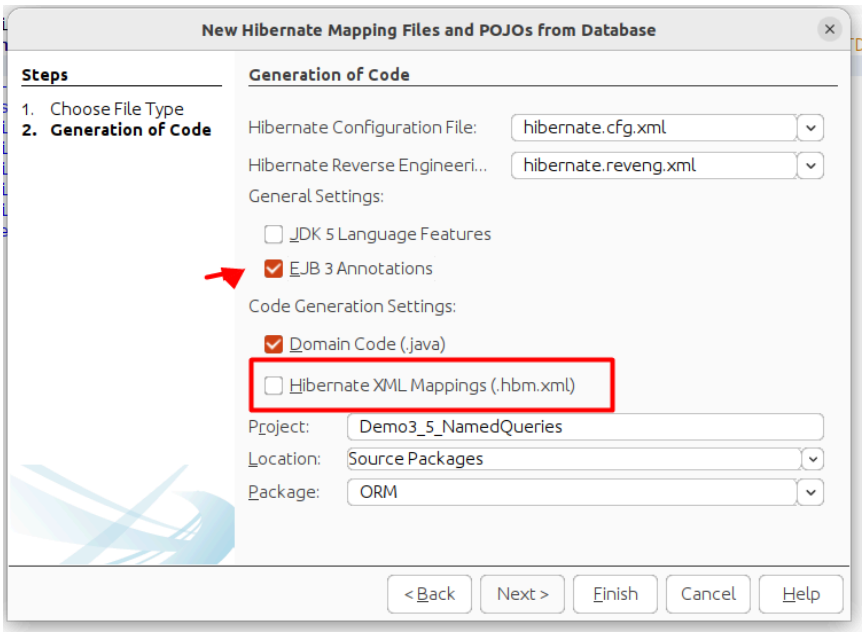
```
(1,'2022-01-30 06:13:12',7.45,2,1),
(2,'2022-01-30 06:13:12',6.5,3,1),
(3,'2022-01-30 06:13:12',4,4,2),
(4,'2022-01-30 06:13:12',8,4,4),
(5,'2022-01-30 06:13:12',10,1,2),
(6,'2022-01-30 06:13:12',3.7,5,2),
(7,'2021-10-10 00:00:00',6.5,1,2),
(8,'2021-12-03 06:13:12',3,5,3);
```

2. Preparar el proyecto Java Maven y los POJOs

Realizamos los pasos necesarios para mapear las tablas en un nuevo proyecto que llamaremos Demo3_5_ConsultasHQL:



1. Modificamos el fichero pom.xml como en otras demos.
2. Creamos el fichero hibernate.cfg.xml para conectar con la nueva base de datos ut3demo5.
3. Creamos el fichero hibernate.reveng.xml para mapear las tablas de esa nueva base de datos.
4. Ejecutamos el wizard para crear los POJOs. En este caso **vamos a utilizar clases mapeadas con anotaciones JPA, no los ficheros hbm.xml**.



5. Comprobamos en hibernate.cfg.xml las etiquetas para mapear las entidades utilizando anotaciones:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "ht
3
4  <hibernate-configuration>
5    <session-factory>
6
7      <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
8      <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ut3demo5?zeroDateTime
9      <property name="hibernate.connection.username">root</property>
10     <property name="hibernate.connection.password">admin</property>
11
12     <property name="hibernate.dialect">org.hibernate.dialect.MySQL5InnoDBDialect</property>
13
14     <property name="hibernate.show_sql">true</property>
15
16     <!-- Mapeo utilizando anotaciones en las clase Java-->
17     <mapping class="ORM.Alumno" />
18     <mapping class="ORM.Examen" />
19     <mapping class="ORM.Modulo" />
20     <mapping class="ORM.Profesor" />
21
22   </session-factory>
23 </hibernate-configuration>
```

¿Por qué no se genera una clase Docencia?

NOTA: *Incluyo estas explicaciones para que entendáis lo que está pasando. Esto no entrará en el examen práctico.*

Hibernate ha detectado la tabla Docencia como una tabla de unión típica para una relación muchos a muchos entre Profesor y Modulo. En este caso, no genera una clase para Docencia porque la trata como una tabla intermedia sin atributos adicionales que deban ser modelados en el código. Esto ocurre cuando la tabla de unión no tiene más columnas aparte de las claves foráneas.

Hibernate solo generará una clase de entidad para la tabla Docencia si esta contiene columnas adicionales (por ejemplo, un atributo como año_académico, horas_clase, etc.). Si Docencia es únicamente una tabla de unión, Hibernate asume que no es necesario representarla como una clase separada.

La relación muchos a muchos entre Profesor y Modulo se modela de la siguiente manera:

- En la clase **`Profesor`**: Hibernate mapea la relación **`Profesor`** ↔ **`Modulo`** como una relación @ManyToMany.

```
@ManyToMany(fetch = FetchType.LAZY)
@JoinTable(
    name = "Docencia",
    catalog = "ut3demo5",
      joinColumns = @JoinColumn(name = "idProfesor", nullable =
false,  updatable = false),
     inverseJoinColumns = @JoinColumn(name = "idModulo", nullable =
false, updatable = false)
)
public Set<Modulo> getModulos() {
    return this.modulos;
}
```

La anotación `@JoinTable` especifica que la tabla `Docencia` actúa como la tabla de unión entre ambas entidades:

- `joinColumns`: Especifica la clave foránea que conecta `Profesor` con la tabla `Docencia` (`idProfesor`).
- `inverseJoinColumns`: Especifica la clave foránea que conecta `Modulo` con la tabla `Docencia` (`idModulo`).

- En la clase **`Modulo`:** La relación se define de forma inversa usando la propiedad `mappedBy`. Esto indica que la tabla `Docencia` ya está mapeada en la clase `Profesor`, y no es necesario redefinir la estructura de la tabla de unión aquí.

```
@ManyToMany(fetch = FetchType.LAZY, mappedBy = "modulos")
  public Set<Profesor> getProfesors() {
      return this.profesors;
  }
```

6. Modificamos los POJOs:
   - Colocamos las anotaciones **en los atributos** en lugar de en los getters (si no, a veces falla).
   - Tenemos que **especificar los tipos genéricos de los Set** que Hibernate genera para modelar relaciones en todas las clases donde aparezcan (Alumno, Modulo, Profesor), en la declaración de los atributos y en los getter y setter de los mismos.
   - Añadimos también el **método toString()** en la clase Alumno, y en cualquiera en la que queramos mostrar información de los objetos creados.

Alumno.java

```
18  @Entity
19  @Table(name = "Alumno",
20          catalog = "ut3demo5"
21  )
22  public class Alumno implements java.io.Serializable {
23
24      @Id
25      @GeneratedValue(strategy = IDENTITY)
26      @Column(name = "idAlumno", unique = true, nullable = false)
27      private Long idAlumno;
28      @Column(name = "apellidos")
29      private String apellidos;
30      @Column(name = "edad")
31      private Integer edad;
32      @Column(name = "nombre")
33      private String nombre;
34      @Column(name = "repetidor")
35      private Boolean repetidor;
36      @OneToMany(fetch = FetchType.LAZY, mappedBy = "alumno")
37      private Set<Examen> examens = new HashSet<>(0);
38
```

Examen.java

```
20  @Entity
21  @Table(name = "Examen",
22          catalog = "ut3demo5"
23  )
24  public class Examen implements java.io.Serializable {
25
26      @Id
27      @GeneratedValue(strategy = IDENTITY)
28      @Column(name = "idExamen", unique = true, nullable = false)
29      private Long idExamen;
30      @ManyToOne(fetch = FetchType.LAZY)
31      @JoinColumn(name = "idAlumno")
32      private Alumno alumno;
33      @ManyToOne(fetch = FetchType.LAZY)
34      @JoinColumn(name = "idModul")
35      private Modulo modulo;
36      @Temporal(TemporalType.TIMESTAMP)
37      @Column(name = "fecha", length = 19)
38      private Date fecha;
39      @Column(name = "nota", precision = 22, scale = 0)
40      private Double nota;
```

Modulo.java

```
19  @Entity
20  @Table(name = "Modulo",
21          catalog = "ut3demo5"
22  )
23  public class Modulo implements java.io.Serializable {
24
25      @Id
26      @GeneratedValue(strategy = IDENTITY)
27      @Column(name = "idModulo", unique = true, nullable = false)
28      private Long idModulo;
29      @Column(name = "nombre")
30      private String nombre;
31      @OneToMany(fetch = FetchType.LAZY, mappedBy = "modulo")
32      private Set<Examen> examens = new HashSet<>(0);
33      @ManyToMany(fetch = FetchType.LAZY, mappedBy = "modulos")
34      private Set<Profesor> profesors = new HashSet<>(0);
```

Profesor.java

```
20  @Entity
21  @Table(name = "Profesor",
22          catalog = "ut3demo5"
23  )
24  public class Profesor implements java.io.Serializable {
25
26      @Id
27      @GeneratedValue(strategy = IDENTITY)
28      @Column(name = "idProfesor", unique = true, nullable = false)
29      private Long idProfesor;
30      @Column(name = "nombre")
31      private String nombre;
32      @ManyToMany(fetch = FetchType.LAZY)
33      @JoinTable(name = "Docencia", catalog = "ut3demo5", joinColumns = {
34          @JoinColumn(name = "idProfesor", nullable = false, updatable = false)}, inverseJoinColumns = {
35          @JoinColumn(name = "idModulo", nullable = false, updatable = false)})
36      private Set<Modulo> modulos = new HashSet<>(0);
```

7. Creamos o copiamos el fichero HibernateUtil.java.
8. Creamos un fichero Principal.java, que usaremos para probar los métodos que implementemos.
9. Creamos un fichero OperacionesHQL.java.

```
public class OperacionesHQL {
    // Atributos
    SessionFactory sf;
    Session s;
    Transaction t;

    public OperacionesHQL() {
        sf = HibernateUtil.getSessionFactory();
    }

    private void abrirSesionYTransaccion() {
        try {
            s = sf.openSession();
            t = s.beginTransaction();
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Error en abrir sesión.");
        }
    }
```

3. Añadir métodos a la clase OperacionesHQL.java

Una vez preparado todo el entorno y creados los ficheros necesarios, crearemos varios métodos en la clase OperacionesHQL.java:

1. Leer y mostrar todos los alumnos
2. Leer un alumno por ID
3. Mostrar nombre y edad de todos los alumnos
4. Consultar nombre y número de exámenes de cada alumno.

   NOTA: la función **size** en HQL se utiliza para obtener la cantidad de elementos de una colección asociada a una entidad. Por ejemplo, en la consulta

   ```
   String hql = "select a.nombre, a.apellidos, size(a.examens) from Alumno a";
   ```

   size (a.examens) devuelve el número de elementos (el tamaño) de la colección examens asociada al objeto Alumno.

5. Obtener nombre y apellidos de alumnos de 24 años que han realizado más de un examen
6. Obtener nombre y apellidos de alumnos de 24 años que han repetido
7. Alumnos repetidores
   a. Obtener lo registros de los alumnos repetidores utilizando una consulta parametrizada
   b. Obtener solo el nombre y apellidos de los alumnos que han repetido
8. Obtener nombre y apellidos de los alumnos de entre 18 y 23 años de edad
9. Obtener todos los alumnos ordenados por edad de forma descendente.
10. Sumar 1 año a los alumnos menores de 20 usando una consulta con parámetros con nombres.
11. Borrar los alumnos repetidores

4. Ejemplo de programa principal

```
package app;

import java.util.logging.Level;
import java.util.logging.Logger;

public class Principal {

    public static void main(String[] args) {
        Logger.getLogger("org.hibernate").setLevel(Level.SEVERE);

        OperacionesHQL o = new OperacionesHQL();

        System.out.println("\n1: Lee todos los alumnos\n");
        o.m1_leerAlumnos();
        System.out.println("\n2: Lee un alumno por ID\n");
        o.m2_leerAlumnoPorID(5L);
        System.out.println("\n3: Nombre y edad de todos los alumnos\n");
        o.m3_mostrarNombreYEdadAlumnos();
        System.out.println("\n4: Consulta cuantos exámenes ha realizado cada alumno\n");
        o.m4_consultarNombreyNumExamenes();
        System.out.println("\n5: Nombre y apellidos alumnos de 24 años que han realizado más de un
examen\n");
        o.m5_alumnos24MasDeUnExamen();
        System.out.println("\n6: Nombre y apellidos alumnos de 24 años que han repetido\n");
        o.m6_alumnos24AniosRep();
        System.out.println("\n7a: Alumnos repetidores\n");
        o.m7a_alumnosRep();
        System.out.println("\n7b: Nombre y apellidos alumnos que han repetido\n");
        o.m7b_nombreYApeAlumnosRep();
        System.out.println("\n8: Nombre y apellidos alumnos entre 18 y 23\n");
        o.m8_nomApeEdadAlumnosEntre18y23();
        System.out.println("\n9: Alumnos ordenados de forma descendente de edad\n");
        o.m9_nomApeEdadAlumnosOrdenEdadDesc();
        System.out.println("\n10: Sumar 1 año a los alumnos menores de 20\n");
        o.m10_sumarUnAnioAlumnosMenor20();
        System.out.println("\n11:Borrar los alumnos repetidores\n");
        o.m11_borrarRepetidores();
        // Comprobamos que ya no hay alumnos repetidores
        System.out.println("\n7a: Alumnos repetidores despues de borrarlos (Debe estar vacio)\n");
        o.m7a_alumnosRep();

        HibernateUtil.cerrarSessionFactory();

    } // Fin del main

} // Fin de la clase Principal
```