Tutoriales Curso JPA

<?xml version="1.0"?><DocumentFileOSQ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"> <GraphicCharterDefinitionId>0</GraphicCharterDefinitionId> <TemplateBaseTypeId>0</TemplateBaseTypeId> <CompanyId>1</CompanyId> <ConfidentialId>0</ConfidentialId> <ConfidentialDescription /> <CountryId>20</CountryId> <PageSizeId>1</PageSizeId> <PageOrientationId>1</PageOrientationId> <PrePrintedStationary>false</PrePrintedStationary> <Project>GEMIS</Project> <Reference>20170124-170150-esgomez</Reference> <TemplateType>3</TemplateType> <CultureId>fr-FR</CultureId> <LanguageId>2</LanguageId> <Customer>Centro de Servicio Valencia</Customer> <DocumentDate>2018-12-12T09:02:22.8300056+01:00</DocumentDate> <Saved>true</Saved> <IsValid>true</IsValid> <FirstPageCover>false</FirstPageCover> <IsNew>false</IsNew> <CurrentVersion>1.00</CurrentVersion> <DocumentType>Solution Requirements Specification</DocumentType> <DocumentTypeId>-1</DocumentTypeId> <Entity /> <HasDistributionList>false</HasDistributionList> <HasForeword>false</HasForeword> <Recipient>Customer Stakeholders (including at least the IT PM and the Stakeholder and User manager)</Recipient> <Title>Spécifications des besoins de la solution - GEMIS</Title> <Status>2</Status> <StatusDescription>Projet</StatusDescription> <SetEdition>false</SetEdition> <SetVersion>false</SetVersion> <TemplateEditor>EN\_eMREQ-SRS</TemplateEditor> <TemplateVersionMajor>1.2</TemplateVersionMajor> <TemplateVersionMinor>4</TemplateVersionMinor> <TemplateYear>2016</TemplateYear> <TemplateState>1</TemplateState></DocumentFileOSQ>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  | Formación Sopra | | | |  |
|  |  |  |
|  |  | JPA básico | | | |  |
|  |  | Version 1.0 Lunes 21 Noviembre 2018 | | | |  |
|  |  |
|  |  |  | | |  |  |

Historial

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Versión** | **Fecha** | **Origen de la actualización** | **Redactado por** | **Validado por** |
| 1.0 | 26/11/2018 | Primera versión del documento | Alba Bermejo Solís  Adrián Colmena Mateos  Emilio Guillem Simón |  |
|  | | | | | |

Contenidos

[1. Tutorial Capítulo 4 4](#_Toc532368672)

[1.1. Creación de tablas e inserción de datos 4](#_Toc532368673)

[1.2. Entidades 5](#_Toc532368674)

[1.3. Queries 9](#_Toc532368675)

[1.3.1. Consultas SELECT 10](#_Toc532368676)

[1.3.2. Consultas WHERE 11](#_Toc532368677)

[1.3.3. Consultas JOIN 12](#_Toc532368678)

[1.3.4. Consultas Group By 13](#_Toc532368679)

[1.3.5. Consultas varias 14](#_Toc532368680)

[1.3.6. Consulta UPDATE 14](#_Toc532368681)

[1.3.7. Consulta DELETE 15](#_Toc532368682)

# Tutorial Capítulo 4

En este capítulo se desarrollará un análisis y uso del lenguaje JP QL (Java Persistance Query Language). Como base de datos se va a utilizar la dada por Oracle, *hr*.

## Creación de tablas e inserción de datos

Se va a usar la base de datos *hr* de Oracle con tres tablas nuevas:

* Tabla **Director**, cuyos campos serán un *number()* **id\_director** que lo identifique de manera única, un *varchar2* **nombre** con el nombre y apellidos del director y un *varchar2* **nacionalidad** con la nacionalidad del director.
* Tabla **Productora**, cuyos campos serán un *number()* **id\_productora** que lo identifique de manera única, un *varchar2* **nombre** con el nombre de la productora y un *varchar2* **pais** con el país al que pertenezca la productora.
* Tabla **Película**, con campos *number()* **id\_pelicula** que será el identificador de la película, un *varchar2* ***titulo***, un *number()* **año** con el año de lanzamiento, un *varchar2* **genero** y dos *number(),* **presupuesto** y **recaudacion** la recaudación y el presupuesto de la película. Además, se debe incluir el director y la productora encargados de la película dentro de la tabla, como claves foráneas.

CREATE TABLE Director(

id\_director number(10) primary key,

nombre varchar2(100),

nacionalidad varchar2(100)

);

CREATE TABLE Productora(

id\_productora number(10) primary key,

nombre varchar2(100),

pais varchar2(100)

);

CREATE TABLE Pelicula(

id\_pelicula number(10) primary key,

titulo varchar2(100),

año number(5),

genero varchar2(100),

director number(10),

productora number(10),

recaudacion number(10, 2),

presupuesto number(10, 2)

);

ALTER TABLE pelicula

add constraint director foreign key(director) references Director(id\_director);

ALTER TABLE pelicula

add constraint productora foreign key(productora) references Productora(id\_productora);

INSERT INTO Director values(1,'Steven Spielberg','Estadounidense');

INSERT INTO Director values(2,'Patty Jenkins','Estadounidense');

INSERT INTO Director values(3,'Julius Avery','Australiano');

INSERT INTO Director values(4,'Guy Ritchie','Británico');

INSERT INTO Director values(5,'Hermanas Wachowski','Estadounidense');

INSERT INTO director VALUES(6,'Santiago Segura', 'Española');

INSERT INTO Productora values(1,'Warner Bros','Estados Unidos');

INSERT INTO Productora values(2,'Paramount Pictures','California');

INSERT INTO Productora values(3,'20th Century Fox','Estados Unidos');

INSERT INTO Pelicula values(1,'Ready Player One: Comienza el juego', 2018, 'Ciencia Ficción', 1, 1, 582, 175);

INSERT INTO Pelicula values(2,'Mujer Maravilla', 2017, 'Ciencia Ficción', 2, 1, 822, 149);

INSERT INTO Pelicula values(3,'Overlord', 2018, 'Terror', 3, 2, 84.7, 38);

INSERT INTO Pelicula values(4,'Snatch, cerdos y diamantes', 2000, 'Acción', 4, 3, 93.6, 6);

INSERT INTO Pelicula values(5,'Pruebas Varias', 1982, 'Comedia', 2, 3, 35.2, 10);

INSERT INTO Pelicula values(6,'Otras Pruebas', 2018, 'Comedia', 4, 2, 94.5, 15);

INSERT INTO Pelicula VALUES(7,'Torrente', 1998, 'Comedia', 6, null, 10.9, 12.7);

INSERT INTO Pelicula values(8,'Pruebas Null', 2018, 'Comedia', null, null, 100, 30);

INSERT INTO Pelicula values(9,'E.T', 1982, 'Ciencia Ficción', 1, 1, 730, 10.5);

INSERT INTO Pelicula values(10,'Matrix', 1999, 'Ciencia Ficción', 5, 1, 464.5, 63);

INSERT INTO Pelicula values(11,'Jurassic Park', 1993, 'Ciencia Ficción', 1, 1, 1029, 63);

## Entidades

A continuación, como en el tema 2, el tutorial necesitará la creación de entidades que puedan ser el reflejo en Java de las tablas creadas en la base de datos. A continuación, se muestra el código de las entidades:

@Entity

@Table(name="DIRECTOR")

**public** **class** Director **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id

@Column(name="ID\_DIRECTOR")

**private** **int** id\_director;

@Column(name="NOMBRE")

**private** String nombre;

@Column(name="NACIONALIDAD")

**private** String nacionalidad;

@OneToMany(mappedBy = "director", cascade = CascadeType.***PERSIST***)

**private** List<Pelicula> peliculas = **new** ArrayList<>();

**public** Director() {

}

**public** Director(**int** id, String nombre, String nacionalidad) {

**super**();

**this**.id\_director = id;

**this**.nombre = nombre;

**this**.nacionalidad = nacionalidad;

}

.

.

.

[Introducir aquí métodos getters y setters]

.

.

.

@Override

**public** **boolean** equals(Object elOtro) {

**if** (elOtro **instanceof** Director) {

Director d = (Director) elOtro;

**return** **this**.id\_director == d.id\_director;

}

**return** **false**;

}

@Override

**public** **int** hashCode() {

**return** (**int**) **this**.id\_director;

}

@Override

**public** String toString() {

**return** String.*format*("id=%d, nombre=%s, nacionalidad=%s", **this**.id\_director,

**this**.nombre, **this**.nacionalidad);

}

//Método para poder imprimir por pantalla solo el nombre del director

**public** String toStringPelicula() {

**return** String.*format*("%s", **this**.nombre);

}

}

@Entity

@Table(name="PRODUCTORA")

**public** **class** Productora **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id

@Column(name="ID\_PRODUCTORA")

**private** **int** id\_productora;

@Column(name="NOMBRE")

**private** String nombre;

@Column(name="PAIS")

**private** String pais;

@OneToMany(mappedBy = "productora", cascade = CascadeType.***PERSIST***)

**private** List<Pelicula> peliculas = **new** ArrayList<>();

**public** Productora() {

}

**public** Productora(**int** id, String nombre, String pais){

**super**();

**this**.id\_productora = id;

**this**.nombre = nombre;

**this**.pais = pais;

}

.

.

.

[Introducir aquí métodos getters y setters]

.

.

.

@Override

**public** **boolean** equals(Object laOtra) {

**if** (laOtra **instanceof** Productora) {

Productora p = (Productora) laOtra;

**return** **this**.id\_productora == p.id\_productora;

}

**return** **false**;

}

@Override

**public** **int** hashCode() {

**return** (**int**) **this**.id\_productora;

}

@Override

**public** String toString() {

**return** String.*format*("id=%d, nombre=%s, pais=%s", **this**.id\_productora, **this**.nombre,

**this**.pais);

}

**public** String toStringPelicula() {

**return** String.*format*("%s",**this**.nombre);

}

}

@Entity

@Table(name="PELICULA")

**public** **class** Pelicula **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id

@Column(name="ID\_PELICULA")

**private** **int** id\_pelicula;

@Column(name="TITULO")

**private** String titulo;

@Column(name="AÑO")

**private** **int** año;

@Column(name="GENERO")

**private** String genero;

@Column(name="RECAUDACION")

**private** **double** recaudacion;

@Column(name="PRESUPUESTO")

**private** **double** presupuesto;

@ManyToOne

@JoinColumn(name="DIRECTOR")

**private** Director director;

@ManyToOne

@JoinColumn(name="PRODUCTORA")

**private** Productora productora;

**public** Pelicula() {

}

**public** Pelicula(**int** id\_pelicula, String titulo, **int** año, String genero, **double**

recaudacion, **double** presupuesto, Director director, Productora productora) {

**super**();

**this**.id\_pelicula = id\_pelicula;

**this**.titulo = titulo;

**this**.año = año;

**this**.genero = genero;

**this**.recaudacion = recaudacion;

**this**.presupuesto = presupuesto;

**this**.director = director;

**this**.productora = productora;

}

.

.

.

[Introducir aquí métodos getters y setters]

.

.

.

@Override

**public** **boolean** equals(Object elOtro) {

**if** (elOtro **instanceof** Pelicula) {

Pelicula l = (Pelicula) elOtro;

**return** **this**.id\_pelicula == l.id\_pelicula;

}

**return** **false**;

}

Para poder imprimir por pantalla el resultado de una consulta que una la entidad Pelicula con las otras dos, pudiendo ser el valor de cualquiera de ellas *null* (sin director o sin productora), se han incluido en el método *toString* las diferentes posibilidades que pueden darse.

@Override

**public** String toString() {

**if**(**this**.director==**null** && **this**.productora==**null**)

**return** String.*format*("id=%s, titulo=%s, año=%d, genero=%s, recaudacion=%.2f,

presupuesto=%.2f, director =null, productora=null,", **this**.id\_pelicula,

**this**.titulo, **this**.año, **this**.genero, **this**.recaudacion, **this**.presupuesto);

**else** **if**(**this**.productora==**null** && **this**.director != **null**)

**return** String.*format*("id=%s, titulo=%s, año=%d, genero=%s, recaudacion=%.2f,

presupuesto=%.2f, director =%s, productora=null,", **this**.id\_pelicula,

**this**.titulo, **this**.año, **this**.genero, **this**.recaudacion, **this**.presupuesto,

**this**.director.toStringPelicula());

**else** **if**(**this**.director==**null** && **this**.productora != **null**)

**return** String.*format*("id=%s, titulo=%s, año=%d, genero=%s, recaudacion=%.2f,

presupuesto=%.2f, director =null, productora=%s,", **this**.id\_pelicula,

**this**.titulo, **this**.año, **this**.genero, **this**.recaudacion, **this**.presupuesto,

**this**.productora.toStringPelicula());

**else**

**return** String.*format*("id=%s, titulo=%s, año=%d, genero=%s, recaudacion=%.2f,

presupuesto=%.2f, director =%s, productora=%s,", **this**.id\_pelicula,

**this**.titulo, **this**.año, **this**.genero, **this**.recaudacion, **this**.presupuesto,

**this**.director.toStringPelicula(),**this**.productora.toStringPelicula());

}

}

Crear a continuación las clases Servicio, una por cada entidad (ServicioDirector, ServicioProductora, ServicioPelicula). También crear la clase Programa para ejecutar el método *main()*.

Todo esto viene explicado en el tema 2 y puede verse el código utilizado para este capítulo en el TutorialJPA.zip.

## Queries

El tutorial emplea sobre todo @NamedQueries, debido a su facilidad de ordenación y utilización más clara. A pesar de ello, @TypedQueries nos da las mismas posibilidades de consulta. Las consultas que nombremos se escribirán en el código de la entidad y las TypedQueries directamente en la clase Servicio de esa entidad. La anotación @NamedQueries y sus @NamedQuery correspondientes se escriben debajo de la anotación @Entity.

### Consultas SELECT

@NamedQueries({

@NamedQuery(name = "pelicula.all", query = "SELECT p FROM Pelicula p "),

.

.

Aquí se introducirá cualquier @NamedQuery que se desee y tenga como objetivo de la consulta, la entidad Pelicula o alguno de sus campos

})

En la clase ServicioPelicula se creará el método que utilice la @NamedQuery anterior:

**public** List<Pelicula> getPeliculasAll(EntityManager em) {

List<Pelicula> peliculas = **new** ArrayList<Pelicula>();

peliculas = em.createNamedQuery("pelicula.all", Pelicula.**class**).getResultList();

**return** peliculas;

}

Este método devolverá una lista de todas las películas que se encuentren en base de datos. Para el programa principal se utiliza uno similar al capítulo 2, utilizando EntityManager(), y *EntityManagerFactory().* El código de la clase Programa será el siguiente para la mayoría de las consultas a realizar:

**public** **class** Programa {

**public** **static** **void** main(String[] args){

EntityManagerFactory emf = Persistence.*createEntityManagerFactory*("cine");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

List<Pelicula> l = ServicioPelicula.*getPeliculasAll*(em);

**for**(Pelicula p:l)

System.***out***.println(p.toString());

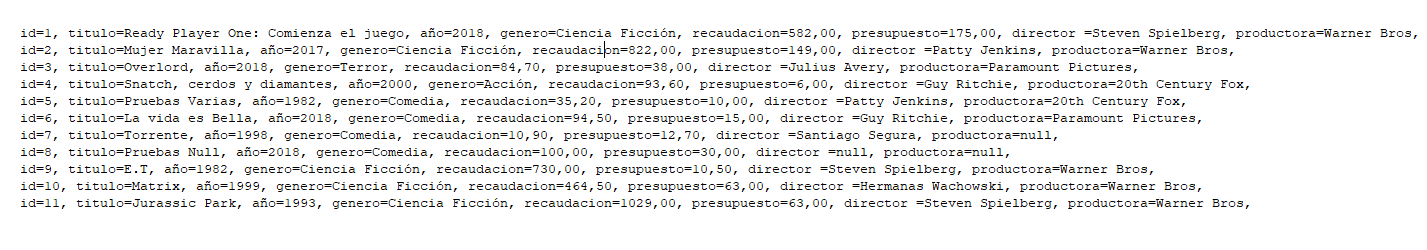
em.getTransaction().commit();

em.close();

emf.close();

}

}

El resultado de esta consulta será:

### Consultas WHERE

@NamedQuery(name = "pelicula.por.genero", query = "SELECT p FROM Pelicula p WHERE p.genero =:genero")

En la clase ServicioPelicula:

**public** **static** List<Pelicula> getPeliculasGenero(EntityManager em) {

String a = "Ciencia Ficción";

List<Pelicula> peliculas = **new** ArrayList<Pelicula>();

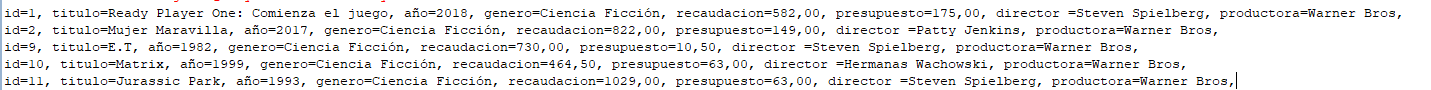
peliculas = em.createNamedQuery("pelicula.por.genero",

Pelicula.**class**).setParameter("genero",a ).getResultList();

**return** peliculas;

}

En el método anterior se crea una variable de tipo *String*, la cual corresponde con un género de película. Aplicando el método *setParameter()* a la consulta podemos realizar consultas dinámicas.

El resultado de la consulta anterior es :

Otro tipo de consulta empleando expresiones de ruta:

@NamedQuery(name = "pelicula.por.director", query = "SELECT p FROM Pelicula p WHERE p.director.nombre =:director")

**public** **static** List<Pelicula> getPeliculasDirector(EntityManager em) {

String a = "Guy Ritchie";

List<Pelicula> peliculas = **new** ArrayList<Pelicula>();

peliculas = em.createNamedQuery("pelicula.por.director",

Pelicula.**class**).setParameter("director",a ).getResultList();

**return** peliculas;

Resultado de la consulta :

#### Distintas consultas con WHERE

Esta consulta devolverá las películas que hayan sido estrenadas a partir del año 2001.

@NamedQuery(name = "pelicula.siglo21", query = "SELECT p FROM Pelicula p WHERE p.año > 2000"),

En esta ocasión se utiliza BETWEEN para consultar qué películas han generado entre 500 y 1000 millones de euros.

@NamedQuery(name = "pelicula.between", query = "SELECT p FROM Pelicula p WHERE p.recaudacion BETWEEN 500 AND 1000"),

La consulta siguiente devolverá las películas pertenecientes a un género cuyo nombre termine con la cadena de texto ‘cció’ seguida por cualquier caracter.

@NamedQuery(name = "pelicula.genero", query = "SELECT p FROM Pelicula p WHERE p.genero LIKE '%cció\_' "),

Esta consulta devolverá la película con mayor recaudación utilizando una subconsulta y la palabra agregada MAX.

@NamedQuery(name = "pelicula.maxrecaudacion", query = "SELECT p FROM Pelicula p WHERE p.recaudacion = (SELECT MAX(pel.recaudacion) FROM Pelicula pel)"),

La consulta siguiente devolverá las películas, donde el director de ellas no sea estadounidense.

@NamedQuery(name = "pelicula.NotInUSA", query = "SELECT p FROM Pelicula p WHERE p.director.nacionalidad NOT IN ('Estadounidense') "),

Esta consulta se devolverán las películas que no tengan productora.

@NamedQuery(name = "pelicula.SinProductora", query = "SELECT p FROM Pelicula p WHERE p.productora IS NULL"),

La siguiente consulta devolverá las películas cuya productora sea distinta a ‘Warner Bros’

@NamedQuery(name = "pelicula.NotWB", query = "SELECT p FROM Pelicula p WHERE NOT EXISTS (SELECT prod FROM p.productora prod WHERE prod.nombre = 'Warner Bros')"),

### Consultas JOIN

@NamedQuery(name = "pelicula.join.director.where", query = "SELECT p.titulo,d.nombre FROM Pelicula p JOIN p.director d WHERE d.nombre ='Steven Spielberg' "),

La consulta anterior une las tablas Pelicula y Director. En este caso devolvemos el título de la película y el nombre del director, es decir, campos de entidades distintas. Por ello el método empleado para devolver la lista cambia ligeramente :

Clase ServicioPelicula :

**public** **static** List<Object[]> getPeliculasJoinDirectorWhere(EntityManager em) {

List<Object[]> peliculas = **new** ArrayList<Object[]>();

peliculas = em.createNamedQuery("pelicula.join.director.where",

Object[].**class**).getResultList();

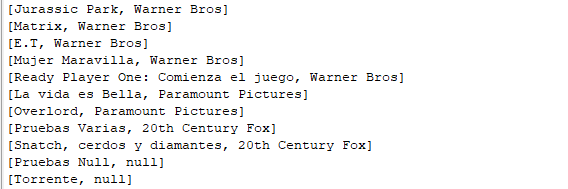
**return** peliculas;

}

La consulta deberá devolver todas las películas cuyo director tenga el nombre de ‘Steven Spielberg’:

La siguiente consulta, une las tablas Pelicula y Productora, pero en este caso devolverá todas las películas, tengan o no productora. El método empleado en la clase ServicioPelicula es prácticamente idéntico al anterior.

@NamedQuery(name = "pelicula.leftjoin.productora", query = "SELECT p.titulo,d.nombre FROM Pelicula p LEFT JOIN p.productora d"),



### Consultas Group By

@NamedQuery(name = "pelicula.groupBy.having", query = "SELECT d.nombre, COUNT(p) FROM Pelicula p LEFT JOIN p.productora d GROUP BY(d.nombre) HAVING COUNT(p)>1")

La consulta anterior devolverá el número de películas que ha realizado cada productora, agrupándolas por nombre, y eliminando aquellas que no hayan hecho más de 1.

Clase ServicioPelicula:

**public** **static** List<Object[]> getPeliculasGroupByHaving(EntityManager em) {

List<Object[]> peliculas = **new** ArrayList<Object[]>();

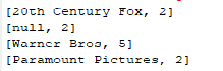
peliculas = em.createNamedQuery("pelicula.groupBy.having",

Object[].**class**).getResultList();

**return** peliculas;

}

Como se puede observar, para cualquier consulta en la que se quiera devolver más de un valor se debe emplear un método que devuelva una lista de *arrays* de *Object*.



### Consultas varias

Otras palabras reservadas permiten poner condiciones distintas a las consultas y poder rescatar aquellos datos que se desee. Algunos ejemplos son los siguientes:

* En esta consulta se seleccionan aquellas películas cuya recaudación es mayor que cualquier de los presupuestos de todas las películas.

@NamedQuery(name = "pelicula.ANY", query = "SELECT p FROM Pelicula p WHERE p.recaudacion < ANY(SELECT pelic.presupuesto FROM Pelicula pelic) ORDER BY p.presupuesto DESC"),

* En este caso se utiliza la palabra CASE para cambiar el género de la película Matrix y Jurassic Park.

@NamedQuery(name = "pelicula.CASE", query = "SELECT p.titulo, "

+ "CASE p.titulo WHEN 'Matrix' THEN 'Filosofica' "

+ "WHEN 'Jurassic Park' THEN 'Dinosaurios' "

+ "ELSE p.genero "

+ "END FROM Pelicula p"),

* En esta consulta se recuperará la media de todos los presupuestos y de las recaudaciones.

@NamedQuery(name = "pelicula.AVG", query = "SELECT AVG(p.presupuesto), AVG(p.recaudacion) FROM Pelicula p"),

### Consulta UPDATE

Para modificar cualquier registro en la base de datos, utilizamos la sentencia UPDATE, donde podemos cambiar el valor de cualquier de las propiedades de una entidad.

@NamedQuery(name = "pelicula.UPD", query = "UPDATE Pelicula p SET p.titulo = 'La vida es Bella' WHERE p.id\_pelicula = 6 "),

El método para la clase ServicioPelicula, difiere de los anteriores siendo del modo:

**public** **static** **void** updatePelicula(EntityManager em) {

em.createNamedQuery("pelicula.UPD").executeUpdate();

}

Se recomienda crear un nuevo registro y luego actualizarlo como se desee.

### Consulta DELETE

Para borrar cualquier registro en la base de datos, utilizamos la sentencia DELETE.

@NamedQuery(name = "pelicula.DEL", query = "DELETE FROM Pelicula p WHERE p.id\_pelicula = 12"),

Este método (de la clase ServicioPelicula), será el mismo que para UPDATE, teniendo en cuenta los cambios realizados en la consulta.

**public** **static** **void** deletePelicula(EntityManager em) {

em.createNamedQuery("pelicula.DEL").executeUpdate();

}

Al igual que con la consulta UPDATE, es recomendable crear un registro en la base de datos antes y luego eliminarlo para probar que la *query* funciona.