LocalMovieDatabase
Desarrollo

ACCDA - Proyecto Hibernate

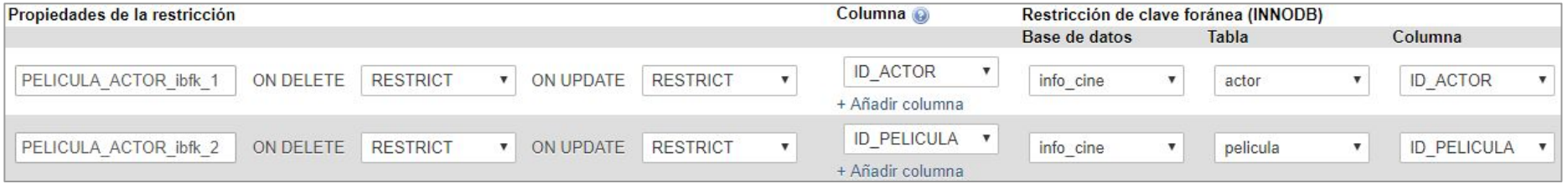
Francisco Rodríguez García

Requisitos mínimos

La BD debe constar de al menos tres tablas relacionadas entre sí

Como puede comprobar en el documento adjunto **clases.pdf**, la Base de Datos cuenta con tres entidades: **Pelicula**, **Actor**, **Director**; cada cual con su tabla, y debido a una relación N:M (muchos a muchos), también hay una cuarta tabla: **PeliculaActor**.

Todas relacionadas con sus respectivas claves foráneas. *Ejemplo en PeliculaActor*:



Deben existir relaciones 1:N (al menos 1)

La relación entre las clases **Director** y **Pelicula** es **1:N**.

Y al hacer la relación N:M entre Pelicula y Actor con la tabla intermedia PeliculaActor, existen dos relaciones **1:N** más → **Pelicula**/**PeliculaActor** y **Actor**/**PeliculaActor**.

Debe existir algún campo autonumérico

Las clases **Pelicula**, **Actor** y **Director** poseen un atributo **id**, relacionados **ID_PELICULA**, **ID_ACTOR** e **ID_DIRECTOR** respectivamente, **campos autonuméricos** en sus tablas de la Base de Datos.

Ejemplo en Pelicula:

```
@Id
@Column(name="ID_PELICULA")
@GeneratedValue(strategy=GenerationType.AUTO)
private int id;
```

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
1	ID_PELICULA	int(11)			No	*Ninguna*		AUTO_INCREMENT

Tipo enumerado en alguna tabla

La clase **Pelicula** tiene dos atributos de tipo enumerado: **pais** y **genero**. Y los guarda en su respectiva tabla en la Base de Datos por el **índice**.

Ejemplo en Pelicula con Pais:

```
@NotNull
@Enumerated(EnumType.ORDINAL)
private Pais pais;
```

6	PAIS	int(3)	No	*Ninguna*

```
public enum Pais {
    ESTADOS_UNIDOS, REINO_UNIDO, INDIA, FRANCIA, ITALIA, ESPANHA, ALEMANIA, JAPON, COREA_DEL_SUR
}
```

Validaciones con Hibernate Validator

Se han incluido validaciones con **Hibernate Validator** en la gran mayoría de atributos de las clases (en los necesarios), utilizando anotaciones de validación como: **@NotNull**, **@Size**(min=n,max=n), **@NotBlank**, **@Digits**(fraction = n, integer = n), **@Min**(value = n), **@Valid** y **@AssertTrue**, configurando algunos mensajes específicos en el fichero **ValidationMessages.properties**.

Ejemplo en Pelicula con tituloEspanha:

```
@NotNull
@Size(min=1,max=100)
@NotBlank
@Column(name="TITULO_ESPANHA")
private String tituloEspanha;
```

Operaciones de inserción en todas las tablas de la BD (al menos 3)

En la clase **Principal** se encuentran los métodos de **inserción** de las tres tablas principales:

- **altaPelicula()**
- **altaDirector()**
- **altaActor()**

Cada método utiliza **guardar()** en su respectivo **DAO** (heredado de GenericDAO).

Operaciones de consultas de todas las tablas de la BD (al menos 3)

Tanto **Película**, como **Actor**, como **Director**, tienen en sus respectivos **DAOs**, el método **localizar(int id)**, que hace una **consulta nativa** devolviendo la entidad en cuestión que tenga un ID específico (si este existe).

Ejemplo en PeliculaDAO:

```
public Pelicula localizar(int id) throws LmdbException {
    Session sesion = HibernateUtil.getSessionFactory().getCurrentSession();

    sesion.beginTransaction();
    Pelicula pelicula = (Pelicula) sesion.get(Pelicula.class, id);

    if (pelicula == null) {
        sesion.getTransaction().rollback();
        throw new LmdbException("No existe la película.");
    }

    sesion.getTransaction().commit();

    return pelicula;
}
```

Consultas con HQL

Las tres clases principales poseen en sus **DAOs** un método para **listar** todo el contenido de sus tablas usando **HQL**, pero, además, el **PeliculaDAO** posee:

- **listarPeliculasPorAnho(String anho)** → Listar películas según un año.
- **listarPeliculasPorPais(int indicePais)** → Listar películas según un país.
- **listarPeliculasPorGenero(int indiceGenero)** → Listar películas según un género.
- **listarPeliculasPorRangoDuracion(int duracionMinima, int duracionMaxima)** → Listar películas según un rango de duración.

Ejemplo en PeliculaDAO con listarPeliculasPorRangoDuracion:

```
createQuery("SELECT p FROM Pelicula p WHERE duracion BETWEEN " + duracionMinima + " AND " + duracionMaxima + "");
```

Operaciones de modificación de al menos una tabla de la BD

En la clase **Principal** se encuentran los métodos de **modificación** de las tres tablas principales:

- **modificarPelicula()**
- **modificarActor()**
- **modificarDirector()**

Cada método utiliza **actualizar()** de su respectivo **DAO** (heredado de GenericDAO).

Operaciones de baja de al menos una tabla de la BD

En la clase **Principal** se encuentran los métodos de **modificación** de las tres tablas principales:

- **bajaPelicula()**
- **bajaActor()**
- **bajaDirector()**

Cada método utiliza **borrar()** de su respectivo **DAO** (heredado de GenericDAO).

Utilización del patrón DAO

El **patrón DAO** (Data Access Object) independiza la aplicación de la forma de acceder a la base de datos, centralizando el código relativo al acceso a datos en clases llamadas DAO, fuera de ellas no debe haber código que acceda a los datos.

Por lo tanto, se han creado las clases:

- **GenericDAO**
  - **PeliculaDAO**
  - **DirectorDAO**
  - **ActorDAO**

Que se encargan del código relativo al acceso a datos.

Gestión de transacciones

La **gestión de transacciones** se hace en las clases **DAO**.

Esta es ayudada por una clase llamada **HibernateUtil**, con utilidades para Hibernate, como crear la factoría de la sesión.

Opciones no obligatorias implementadas

Existencia de una relación N:M implementada como dos relaciones y 1:N

Existe una relación **N:M** entre **Pelicula** y **Actor**, implementada como **dos** relaciones **1:N** mediante una nueva clase de unión: **PeliculaActor**.

Existencia de relación 1:N ordenada

La relación **1:N** entre **Director** y **Pelicula** es **ordenada**.

```
@OneToMany(cascade=CascadeType.ALL)
@JoinColumn(name="ID_DIRECTOR")
private List<Pelicula> peliculas;
```

Tipo fecha en alguna tabla

La **tabla pelicula** tiene el atributo **anho** como **Date**, al igual que la **clase Pelicula** con su atributo **anho**.

4	**ANHO**	date	No	*Ninguna*

```
@NotNull
@Column(name="ANHO")
private Date anho;
```

Métodos Java de validación

La clase **Pelicula** tiene un **método Java para la validación** del año mediante **Hibernate Validator**.

```
@AssertTrue(message="El año no es válido")
private boolean isAnhoPelicula() {
    boolean esValido;
    String expresionRegular = "\\s\\d{4}";

    if (Pattern.matches(expresionRegular, this.anho.toString())) {
        esValido = true;

    } else {
        esValido = false;
    }

    return esValido;
}
```

Configuración de Cascade

Configuración de Cascate

Se ha optado por **CascadeType.ALL** en todas las relaciones.

Ya que, por ejemplo, una película no podría existir sin un director, por tanto, si se elimina un director, no existen sus películas.

Dificultades encontradas en el desarrollo

Dificultades encontradas en el desarrollo

- Parsear y dar el formato adecuado al año utilizando Date, así como hacer el patrón necesario para su validación.
- Mostrar los listados de las diferentes entidades en una tabla por consola formateada automáticamente según la celda más ancha de cada columna.
- Trabajar con una cantidad considerable de clases y métodos respecto de lo que estamos acostumbrados con un proyecto de una relación normal (por ejemplo, solo la clase Principal tiene más de 1800 líneas).

Decisiones de optimización

Decisiones de optimización

Para una mejor optimización del desarrollo y la legibilidad:

- Las clases se han dividido en paquetes según su función.
- Se ha modularizado para reutilizar código.
- Se han diferenciado las secciones en la clase Principal con comentarios llamativos para una mayor claridad visual.
- Se han creado el mayor número de constantes posibles para facilitar cambios.
- Se han declarado los nombres de los métodos, de las variables y constantes de la forma más descriptiva posible.