



CFP César Manrique

Introducción a Hibernate

**CFGS DAW – Desarrollo web en entorno
servidor**



ÍNDICE DE CONTENIDOS

➤ Introducción a Hibernate



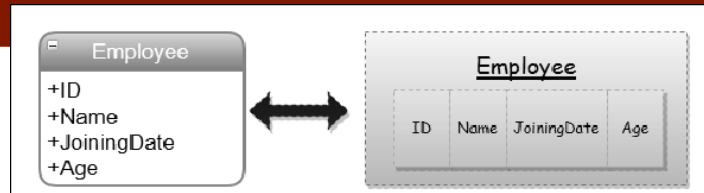


Hibernate 4 ORM





Introducción a Hibernate



- Hibernate es una solución ORM para entornos Java
- El acrónimo ORM (Object Relational Mapping) implica la transformación de un modelo, en forma de colecciones de objetos, a una representación soportada por un modelo relacional (o similar). Esta transformación se realiza en ambos sentidos
- Además de “mapear” tablas de un modelo E/R en objetos, Hibernate proporciona mejoras en operaciones de consulta y manipulación de datos (DML)
- Hibernate está descompuesto en un amplio número de módulos, con objeto de eliminar acoplamientos innecesarios con módulos no necesarios.
- **hibernate-core** define la base ORM , así como APIs y SPIs de integración (SPI: *Service Provider Interface*)



Dependencias

- Añadimos dependencia Maven a hibernate-core 4.3.11.final

Add Dependency

Group ID:

Artifact ID:

Version: Scope:

Type: Classifier:

Search **Open Projects** **Dependency Management**

Query:
(coordinate, class name, project name...)

Search Results:

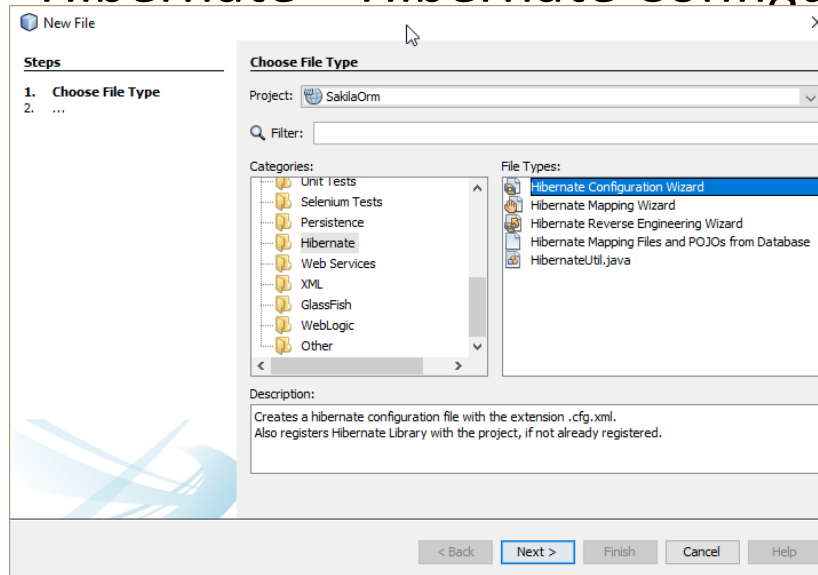
	5.0.0.CR3 [jar] - central
	5.0.0.CR2 [jar] - central
	5.0.0.CR1 [jar] - central
	5.0.0.Beta2 [jar] - central
	5.0.0.Beta1 [jar] - central
	4.3.11.Final [jar] - central
	4.3.11.Final [jar] - local
	4.3.10.Final [jar] - central
	4.3.9.Final [jar] - central
	4.3.8.Final [jar] - central
	4.3.7.Final [jar] - central

Add **Cancel**

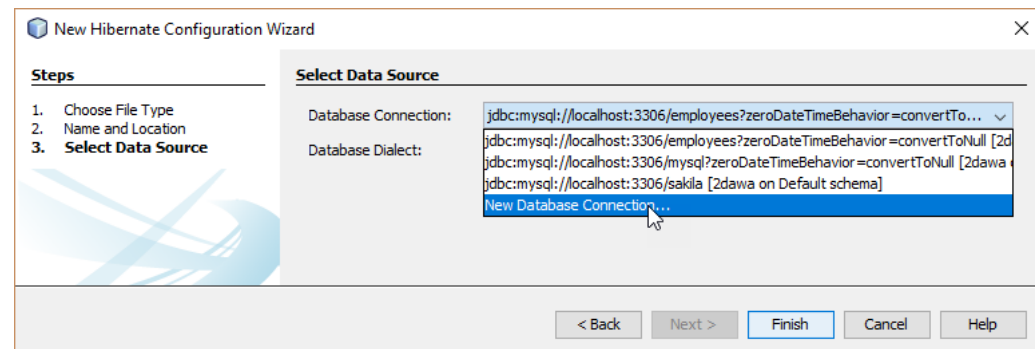


Crear fichero de configuración de Hibernate

- Creamos el hibernate.cfg.xml (irá a “Other Sources”)
 - New > Hibernate > Hibernate Configuration Wizard



- Select Data Source
 - New Database Connection





Introducir coordenadas para la nueva conexión

- Introducimos datos para conexión a sakila
 - Host: localhost
 - Port: 3306
 - Database: sakila
 - User: 2dawa
 - Password: 2dawA2!06
 - Quitamos parámetro de la JDBC URL

The screenshot shows the 'New Connection Wizard' dialog box, specifically the 'Customize Connection' step. The fields are filled with the following information:

- Driver Name: MySQL (Connector/J driver)
- Host: localhost
- Port: 3306
- Database: sakila
- User Name: 2dawa
- Password: 2dawA2!06 (masked with dots)
- ☒ Remember password
- Buttons: Connection Properties, Test Connection
- JDBC URL: jdbc:mysql://localhost:3306/sakila

At the bottom, there are navigation buttons: < Back, Next > (highlighted), Finish, Cancel, and Help.




Resultado

- Se generaría un hibernate.cfg.xml como este:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/sakila</property>
    <property name="hibernate.connection.username">2dawa</property>
    <property name="hibernate.connection.password">2dawA2!06</property>
  </session-factory>
</hibernate-configuration>
```

- Luego, en ejecución, conectaremos con datasource

```
<property name="hibernate.connection.datasource">java:comp/env/jdbc/sakila</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="hibernate.current_session_context_class">thread</property>
```

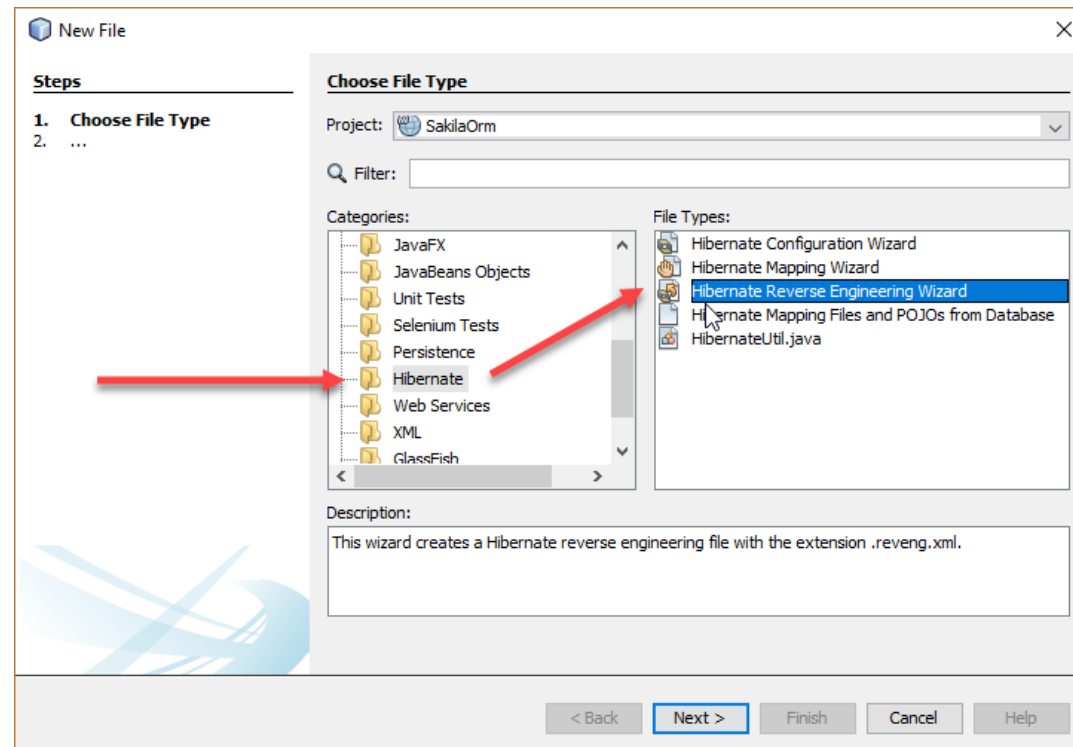




Crear fichero de ingeniería inversa

- En nuestro ejemplo vamos a seguir una aproximación **Database First (o bottom-up)**. Quiere decir que la base de datos ya existe, vamos a adaptar el modelo de objetos al modelo E/R (entidad-relación) ya existente
- Seleccionamos

Hibernate > Hibernate Reverse Engineering Wizard

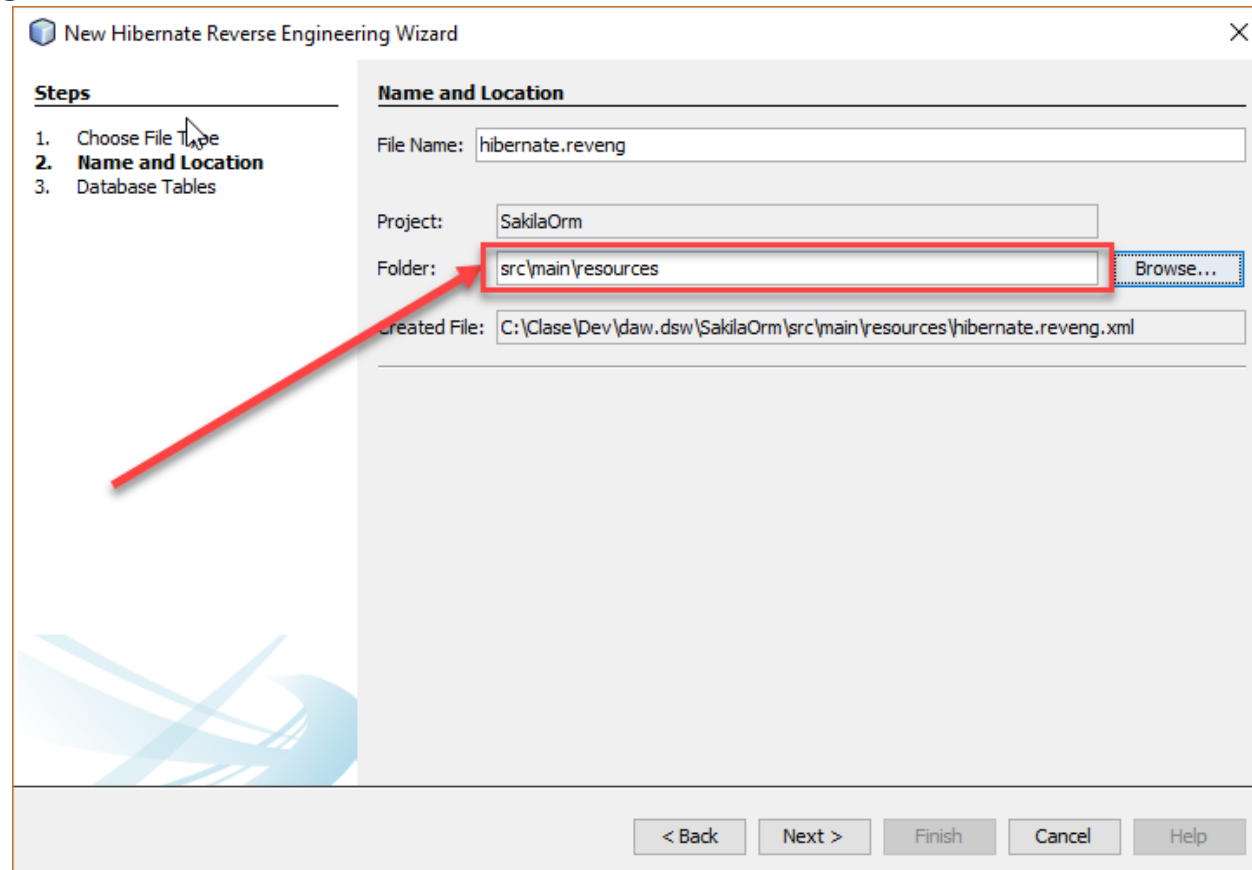




Reverse Engineering

- Cuidado, seleccionad una carpeta para el fichero que va a ser generado. En nuestro caso **hibernate.reveng**

Si no seleccionamos
carpeta no lo crea

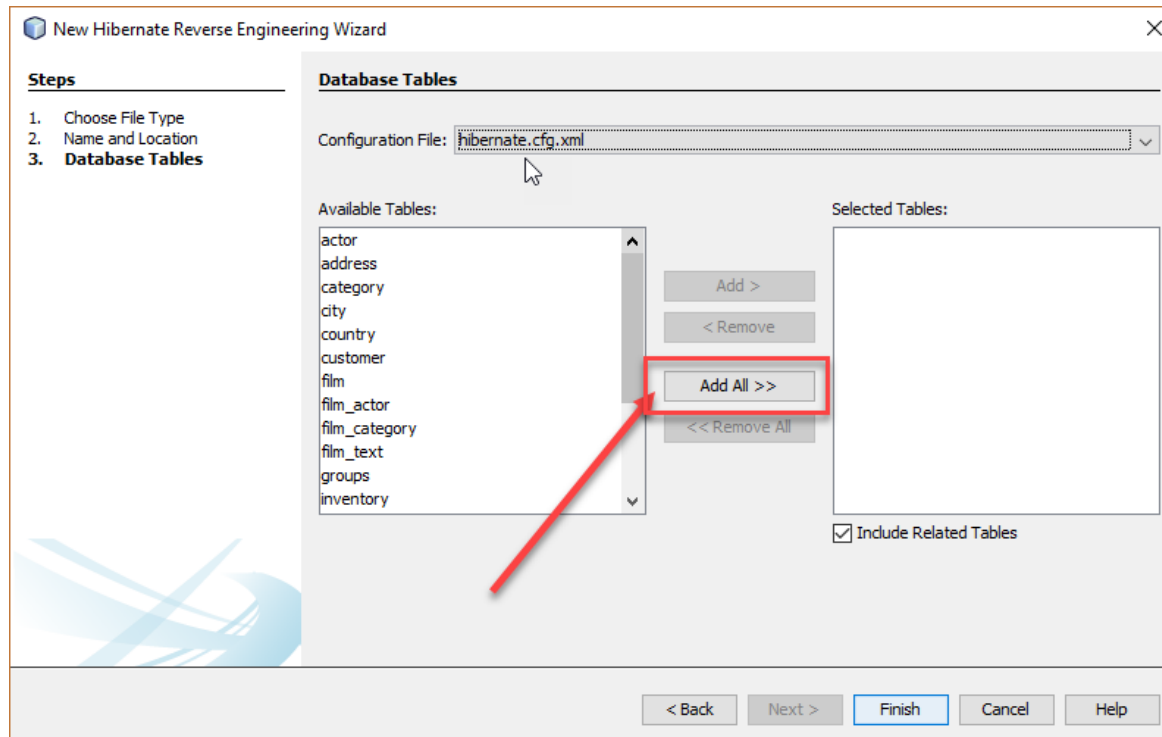




Reverse Engineering

- Seleccionamos el fichero de configuración (**hibernate.cfg.xml**) y seleccionamos las tablas deseadas

Si está marcado “Include Related Tables” se traerá también las que estén relacionadas por Foreign Key





Reverse Engineering

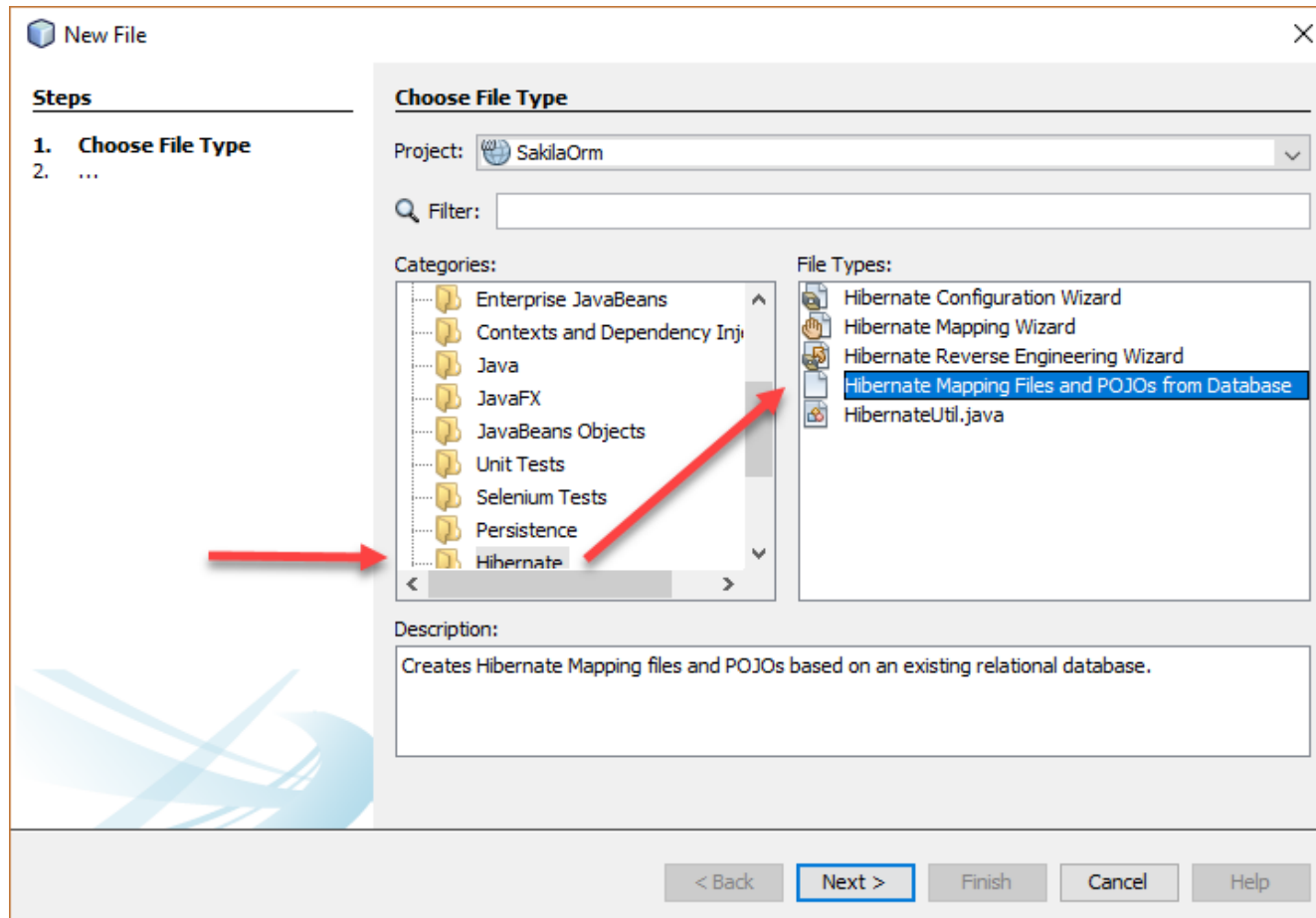
- El resultado es el fichero **hibernate.reveng.xml**
 - Aquí aparecerán las tablas que luego va a crear el asistente de generación de POJOs
 - Se crearán clases Java en base a lo que aparezca aquí
 - Los nombres de clases, que se vayan a generar, podemos personalizarlos en este fichero
- Luego veremos cómo

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-reverse-engineering PUBLIC "-//H
<hibernate-reverse-engineering>
  <schema-selection match-catalog="sakila"/>
  <table-filter match-name="film"/>
  <table-filter match-name="film_category"/>
  <table-filter match-name="address"/>
  <table-filter match-name="staff"/>
  <table-filter match-name="rental"/>
  <table-filter match-name="users_groups"/>
  <table-filter match-name="customer"/>
  <table-filter match-name="groups"/>
  <table-filter match-name="inventory"/>
  <table-filter match-name="film_text"/>
  <table-filter match-name="actor"/>
  <table-filter match-name="users"/>
  <table-filter match-name="store"/>
  <table-filter match-name="film_actor"/>
  <table-filter match-name="city"/>
  <table-filter match-name="country"/>
  <table-filter match-name="language"/>
  <table-filter match-name="payment"/>
  <table-filter match-name="category"/>
</hibernate-reverse-engineering>
```



Generando POJOs para entidades del modelo E/R

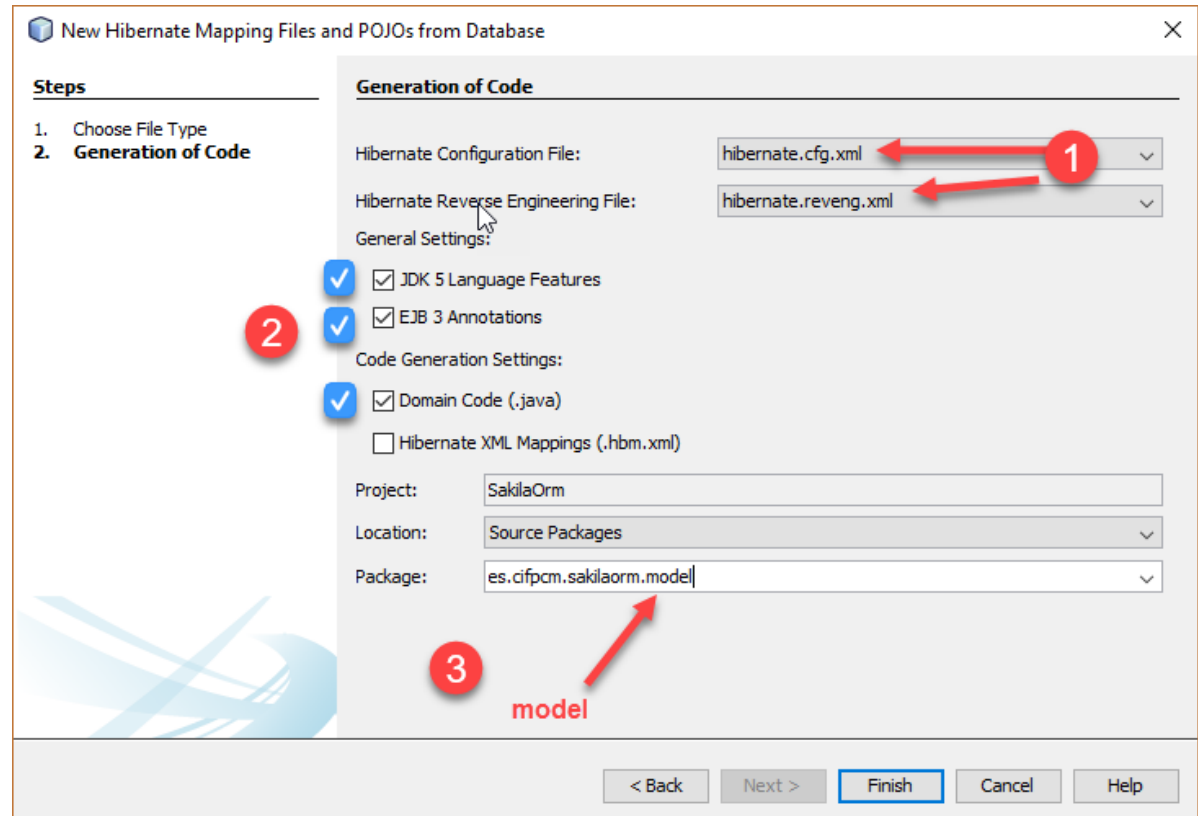
- Seleccionamos **Hibernate > Hibernate Mapping Files and POJOs from Database**






Generar POJOs de entidades de BD

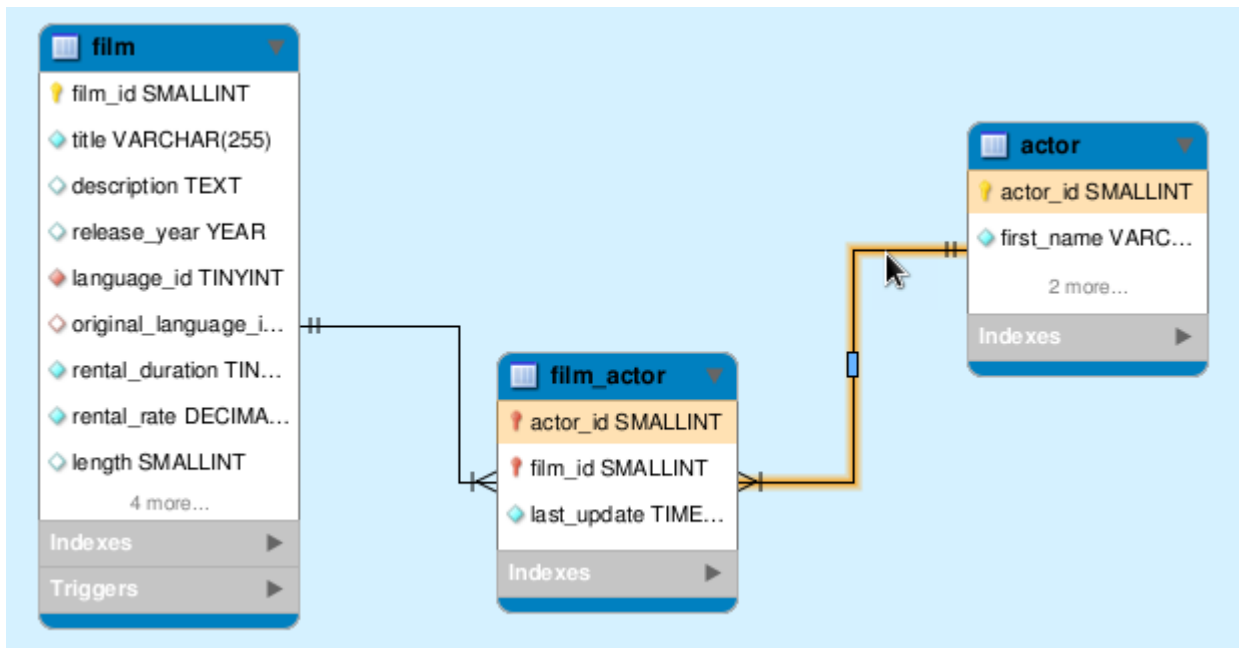
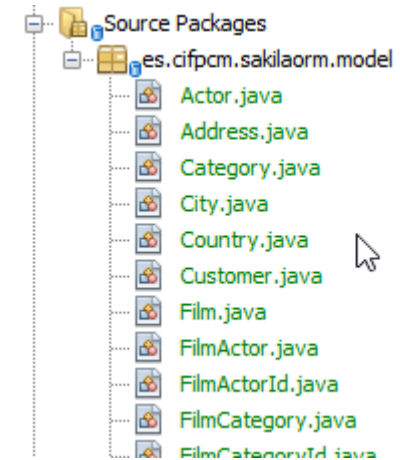
- 1) Seleccionamos el fichero de configuración e ingeniería inversa generados anteriormente
- 2) Marcamos las opciones indicadas (XML lo desmarcamos)
- 3) Indicamos que las clases generadas se creen en el subpaquete model





Clases generadas por Hibernate

- El asistente nos ha generado las clases Java en el paquete indicado 
- Vamos a analizar el código generado para las entidades de una relación N:M





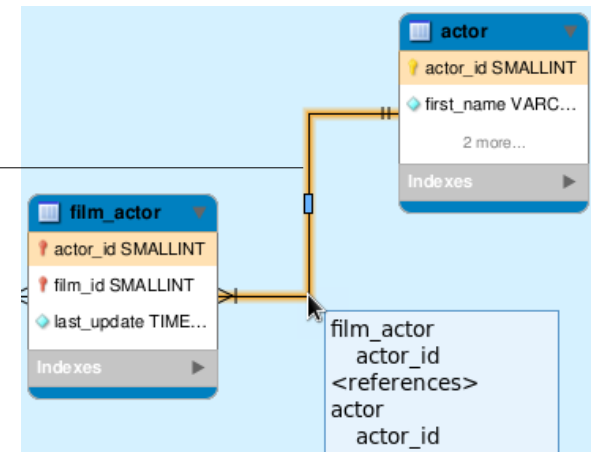
Clase Actor: Mapeo de entidad E/R

- Podemos ver la clase anotada con `@Entity` y `@Table`
- `@Entity` marca la clase como asociada a modelo E/R y `@Table` asocia la clase a una entidad (tabla) de Base de datos

```
@Entity
@Table(name="actor"
      ,catalog="sakila"
)
public class Actor implements java.io.Serializable {

    private Short actorId;
    private String firstName;
    private String lastName;
    private Date lastUpdate;
    private Set<FilmActor> filmActors = new HashSet<FilmActor>(0);

    public Actor() {
    }
}
```



- Vemos también que Hibernate ha creado un `Set<FilmActor>` esto es así porque ha detectado una relación 1:N de Actor a FilmActor



Clase Actor: Mapeo de columnas

- @Id identifica la clave primaria PK de la entidad
- @GeneratedValue indica que esa columna la genera la BD, siguiendo una estrategia autoincremental
- @Column asocia una propiedad con un nombre de columna de la tabla

```
@Id
@GeneratedValue(strategy = IDENTITY)
@Column(name = "actor_id", unique = true, nullable = false)
public Short getActorId() {
    return this.actorId;
}

public void setActorId(Short actorId) {
    this.actorId = actorId;
}

@Column(name = "first_name", nullable = false, length = 45)
public String getFirstName() {
    return this.firstName;
}
```



Clase Actor: Asociaciones

- En el ejemplo de código vemos que Hibernate mantiene una colección de objetos FilmActor asociada a Actor
- Como hemos visto antes, esto es así, porque existía una relación actor 1:N film_actor
- El atributo **fetch** indica Lazy. Esto quiere decir que Hibernate no se trae la colección de FilmActor asociada cuando recupere un actor. Estos datos se traerán cuando se necesiten (cuando se acceda al getter)
- Hibernate puede obtener automáticamente esos datos, siempre que el **objeto (proxy) esté vinculado a una sesión activa**


```
@OneToMany(fetch = FetchType.LAZY, mappedBy = "actor")  
public Set<FilmActor> getFilmActors() {  
    return this.filmActors;  
}
```

Personalizando la generación

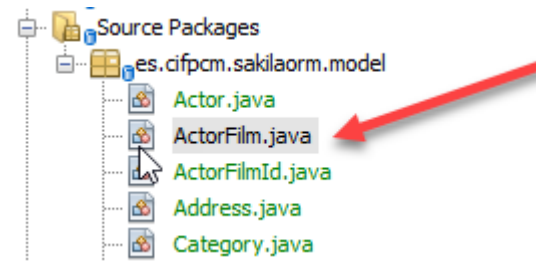
- Puedo especificar elementos **<table>** en hibernate-reveng.xml para personalizar la generación del POJO

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-reverse-engineering PUBLIC "-//Hibernate/Hibernate Reverse Engineering DT
<hibernate-reverse-engineering>
  <schema-selection match-catalog="sakila"/>
  <table-filter match-name="film"/>
  <table-filter match-name="film_category"/>
  <table-filter match-name="film_actor"/>

  <table name="film_actor" catalog="sakila" class="es.cifpcm.sakilaorm.model.ActorFilm"/>
</hibernate-reverse-engineering>
```



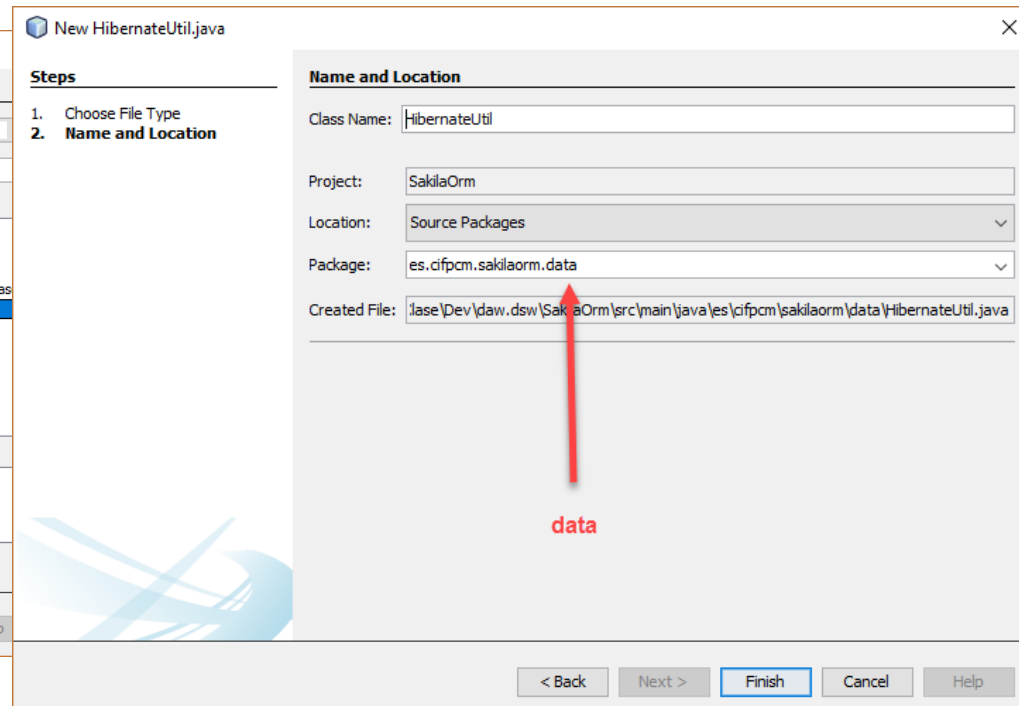
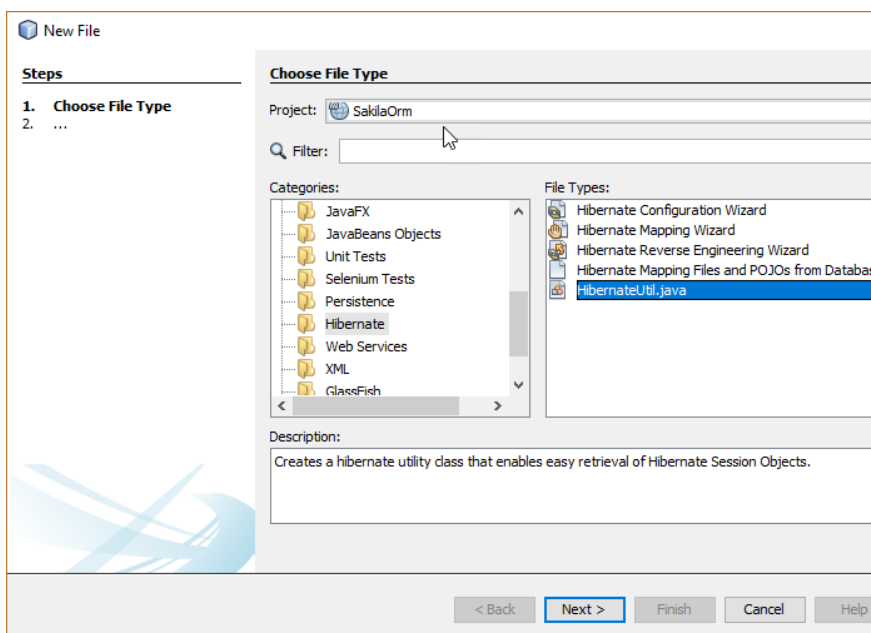
- Comprobamos que la clase ahora se llama ActorFilm (se pueden personalizar muchas más cosas)





Hibernate Native Bootstrapping

- Para el bootstrap de Hibernate vamos a utilizar una clase HibernateUtil
- https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html#bootstrap-nativeorac





Cambios en Hibernate Util

- La misión de esta clase es devolver la factoría de sesiones: **SessionFactory**
- El código auto-generado es obsoleto, vamos a modificar el código *deprecated*

```
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.SessionFactory;

/**
 * Hibernate Utility class with a convenient method to get Session Factory
 * object.
 *
 * @author Fer
 */
public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from standard (hibernate.
            // config file.
            sessionFactory = new AnnotationConfiguration().configure();
        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed. " + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Sigue estando deprecated
El método buildSessionFactory sin argumentos
Pero ya passsooo

```
public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from standard (hibernate.cfg.xml)
            // config file.
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

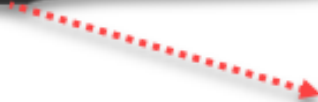
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```



Ejemplos de código

- Consultas vía CriteriaQuery
- Lenguaje tipo Fluent donde vamos encadenando llamadas

```
private List<Actor> fetchActors() {  
    List<Actor> aList = new ArrayList<>();  
    Session session = HibernateUtil.getSessionFactory().openSession();  
    try {  
        session.beginTransaction();  
  
        aList.addAll(session.createCriteria(Actor.class).list());  
  
        session.getTransaction().commit();  
    } catch (Exception ex) {  
        session.getTransaction().rollback();  
    } finally {  
        session.close();  
    }  
    return aList;  
}
```



Criteria query



Ejemplos de código

- Ejemplo HQL
- Lenguaje parecido a SQL pero más orientado a Objetos
- Indicamos los nombres de clase de los POJOs, no las tablas
- Las relaciones las realizamos a través de las propiedades

```
final String query = "select a from Actor a "  
    + "inner join fetch a.actorFilms af "  
    + "inner join fetch af.film "  
    + "where a.actorId=:actorId";  
  
List<Film> afList = new ArrayList<>();  
  
Session session = HibernateUtil.getSessionFactory().openSession();  
try {  
    session.beginTransaction();  
  
    Actor a  
        = (Actor) session.createQuery(query).  
            setShort("actorId", actorId).list().get(0);
```



Operaciones de actualización

- Muy simples

```
Actor actor = new Actor();
actor.setFirstName(getFirstName());
actor.setLastName(getLastName());
Session session = HibernateUtil.getSessionFactory().openSession();
try {
    session.beginTransaction();
    session.save(actor);
    session.getTransaction().commit();
    this.id = actor.getActorId();
    return "actorDetail";
} catch (Exception ex) {
    session.getTransaction().rollback();
    return null;
} finally {
    session.close();
}
```




Links

- Curso de Hibernate con Spring
<http://www.cursohibernate.es/doku.php>



PREGUNTAS

