TP Hibernate - BD «Film»

1 Objectif

Le but de ce tp est de vous familiariser avec HIBERNATE pour les échanges d'information entre une application JAVA et une base de données. Vous découvrir **`Hibernate tools`** un plugin pour utiliser HIBERNATE sous ECLIPSE .

2 Avant de commencer

2.1 Méthodologie de développement

Pour effectuer votre développement et vos tests vous utiliserez les outils suivants :

- L'outil de développement ECLIPSE étendu par le plugin WTP (Web Tools Project).
- Une base de données MYSQL installée par les administrateurs sur le serveur **`anteros`** de l'IFSIC.
- HIBERNATE un ensemble d'outils pour le «mapping» objet JAVA vers une base de données relationnelle.
- Le Plugin «**`Hibernate tools`**» fourni par JBOSS en complément de WTP pour l'utilisation d'HIBERNATE.

2.2 Base de donnée MYSQL sur le serveur `anteros` de l'IFSIC

Accéder au serveur **`anteros`** de l'IFSIC à l'aide d'un navigateur à l'URL : **`http://anteros.ifsic.univ-rennes1.fr`** :

FIGURE 1. Page d'accueil du serveur `anteros`

Si vous n'avez pas encore de base de données sous le serveur **`anteros`** laissez vous guider par les instructions de la page d'accueil du serveur pour créer une

base de données MySQL. Donnez lui un mot de passe que vous utiliserez plus loin pour accéder à votre base. La base ainsi créée n'a pas de table pour l'instant, mais pour ce TP, nous vous fournissons dans les ressources du TP, une classe JAVA qui, après adaptation, effectuera la création de tables dans votre base. Cette tâche vous sera demandé plus loin.

Remarque La base que vous venez de créer vous appartient en propre, elle peut être utilisée dans plusieurs projets ou TP de votre cursus tout au long de l'année. Le menu MySQL de la page d'accueil d'**`anteros`** vous permet de revenir ultérieurement gérer cette base. Par cette page vous pourrez changer le mot de passe si vous l'avez oublié. Vous pourrez aussi visualiser le contenu des tables en créer, en effacer. Nous vous encourageons à faire périodiquement cette tâche d'administration.

2.3 HIBERNATE

HIBERNATE est un ensemble d'outils facilitant le développement d'applications JAVA utilisant un SGBD relationnel pour la persistence de ses données. HIBERNATE simplifie à l'extrême la conception et la programmation des échanges entre l'application et la base de données. HIBERNATE maintient automatiquement une cohérence entre les valeurs manipulées dynamiquement par l'application et celles présentes dans la base et prend en charge la gestion des pools de connections et des requètes.

2.3.1 Configuration d'HIBERNATE

Il y a trois manières d'effectuer la configuration d'HIBERNATE pour une application donnée :

1. Par programme
2. Par un fichier de propriétés **`hibernate.properties`**
3. Par un document XML **`hibernate.cfg.xml`**

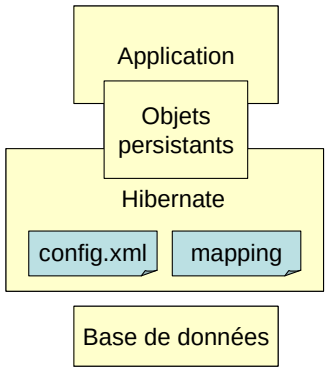


FIGURE 2. HIBERNATE **et la persistance de données**

C'est cette troisième solution que nous allons employer.

2.3.2 Emplacement du fichier de configuration

HIBERNATE introspecte dynamiquement le fichier de configuration. Sous ECLIPSE il est recommander de déposer ce fichier de configuration avec les sources d'un projet JAVA. Ainsi placé avec les sources, ce fichier sera copié automatiquement par ECLIPSE dans le répertoire des classes. Il sera ainsi visible dans le **`classpath`** de votre projet et HIBERNATE pourra l'accèder dynamiquement.

2.4 Ressources pour ce tp

Nous vous fournissons pour ce TP des ressources réparties en deux répertoires, un répertoire **`lib`** contenant des librairies JAVA et un répertoire **`src`** contenant des sources JAVA :

- la librairie **`hibernate3.jar`** ainsi que les (nombreuses) librairies tiers utilisées par HIBERNATE.
- un driver JDBC pour la base MYSQL, il se trouve dans la librairie **`mysql-connector-java-5.0.4-bin.jar`**.
- quelques classes JAVA dont une classe **`HibernateUtil`** elle même fournie par JBOSS avec la librairie HIBERNATE. Cette classe permet de créer des sessions avec une seule et unique instance de la classe **`SessionFactory`**.
- la classe **`InitFilm`** qui va vous permettre l'initialisation de la base **`film`.**

Voici une photo des ressources founies pour ce TP. Toutes les librairies se trouvent dans le répertoire **`lib`**.

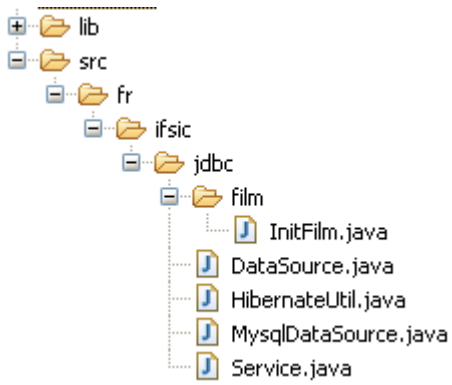


FIGURE 3. Ressources pour le tp

2.5 Création des tables de la base film

Question 1. Initialisation de la base

Créez un projet JAVA. *Importer les ressources à partir du fichier* **`ressources.zip`** *fourni pour ce* TP. *Ajoutez à votre* **`classpath`** *toutes les librairies présentes dans le répertoire* **`lib`** *de votre projet. Modifiez le constructeur de la classe* **`InitFilm`** *pour l'adapter à vos besoins :*

```
public InitFilm() {
```

```
        super("votre identifiant", "votre mot de passe");
    }
```

Une fois la classe **`InitFilm`** *modifiée par vos soins exécutez cette classe. Les tables de la base* **`film`** *vont être créées automatiquement dans votre base* MYSQL *sur* **`anteros`**. *Vérifiez votre base par le programme d'administration d'*`anteros`.

3 Ingénierie inverse à partir d'une base existante

HIBERNATE utilise la librairie JDBC pour accéder aux données et aux méta données d'une base de données relationnelle. Par introspection de la base, HIBERNATE peut générer des classes JAVA correspondant au modèle de la base. Un document XML **`reveng.xml`** doit être préparé qui décrit ce processus d'ingénierie inverse. A partir de ce document, il est ensuite possible de générer les classes correspondant au modèle de votre base de données. C'est ce que vous allez faire maintenant.

3.1 Configuration d'HIBERNATE

Du point de vue d'HIBERNATE, la première tâche consiste à créer le fichier de configuration **`hibernate.cfg.xml`** qui permet à HIBERNATE de gérer les connections à la base que vous venez de créer.

Question 2. Création du fichier de configuration

En utilisant l'assistant de création de fichiers de configuration fourni par le plugin **`Hibernate Tools`** *créez le fichier de configuration. Procédez comme suit :*

Lancez l'assistant de création ce fichier de configuration par le menu :

`File>New>Other ...>Hibernate>Hibernate Configuration File`

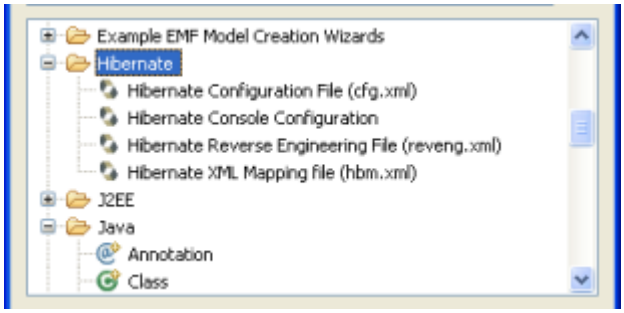


FIGURE 4. Les assistants de création fournis par HIBERNATE tools

Dans la page qui s'ouvre sélectionnez ou confirmer l'emplacement du fichier de configuration. Il doit se trouver à la racine de vos sources, c'est à dire dans le répertoire **`src`**.

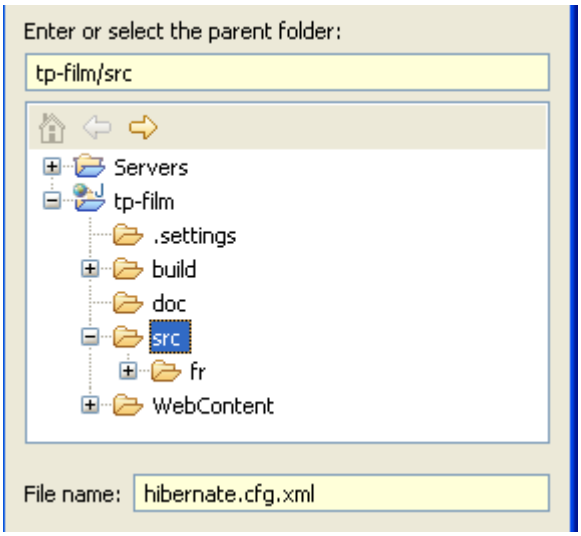


FIGURE 5. Choisir un emplacement et un nom pour le fichier de configuration

Cliquez sur le bouton **`Next`**, dans la nouvelle page choisir le dialecte de base de données **`MySQL`** et remplir les 4 champs **`Driver Class`**, **`Connection URL`**, **`Username`** et **`Password`** (utilisez votre identifiant sésame et votre mot de passe) :

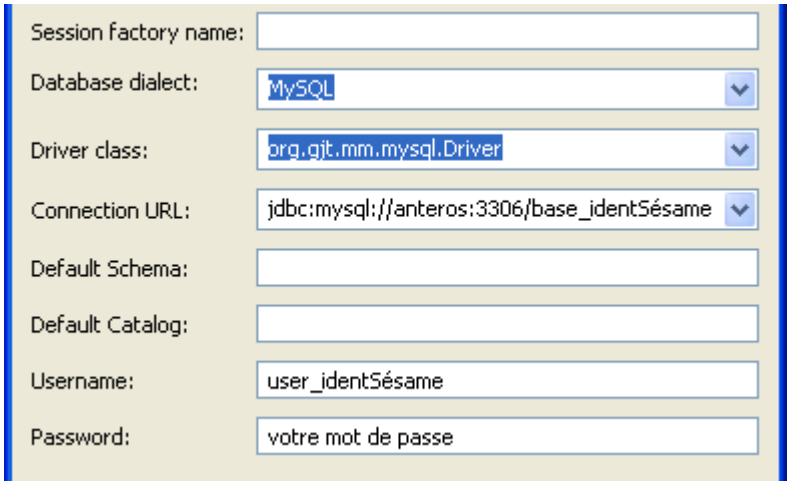


FIGURE 6. Informations de bases pour le fichier de configuration

Le fichier de configuration **`hibernate.cfg.xml`** ainsi créé doit ressembler à celui-ci :

```
<hibernate-configuration>
  <session-factory>
    <property
     name="hibernate.connection.driver_class">org.gjt.mm.mysql.Dr
     iver</property>
    <property name="hibernate.connection.password">xxx</property>
    <property name="hibernate.connection.url">jdbc:mysql://
     anteros:3306/base_id</property>
    <property name="hibernate.connection.username">user_id</
     property>
```

```
    <property
     name="hibernate.dialect">org.hibernate.dialect.MySQLDialect<
     /property>
  </session-factory>
</hibernate-configuration>
```

Il s'agit d'un fichier de configuration minimum que vous auriez pu construire à la main. Pour l'instant la seule chose qu'il décrit est la manière dont HIBERNATE doit se connecter à votre base et c'est la seule chose dont nous avons besoin pour effectuer le processus d'ingénierie inverse.

Votre fichier de configuration **`hibernate.cfg.xml`** est maintenant créé, vérifiez qu'il se trouve au bon endroit, à la racine de vos sources.

3.2 Création d'une configuration de console.

Le plugin **`Hibernate tools`** possède des *configurations de console* ; chaque configuration permet d'accéder à un ensemble de tables dans une base, de les manipuler ainsi que de lancer, si on le désire, le processus d'ingénierie inverse d'HIBERNATE. Par ce processus il est possible de produire automatiquement les classes correspondant à un modèle table de la base de données. C'est ce que nous allons faire à partir de la base **`film`**.

Question 3. Mise en place d'une configuration de console

Créez une configuration de console en procédant comme suit :

Sélectionnez le fichier de configuration que vous venez de créer puis lancez l'assistant de création de configuration de console. Vous trouvez cet assistant par le menu :

`File>New>Other ...>Hibernate>Hibernate Console Configuration`

Quand l'assistant de création démarre il utilise le fichier de configuration sélectionné pour effectuer le meilleur choix de configuration adapté à votre projet.

FIGURE 7. L'assistant de création de configuration de console.

Dans cette page deux choses sont importantes le nom de projet relatif à cette configuration de console et le nom de fichier de configuration. C'est là qu'HIBERNATE trouve les informations pour sa connection à votre base. Vérifier le nom du fichier de configuration. Ouvrez l'onglet **`Classpath`**, vérifiez que la case **`Include default classpath from project`** est bien cochée. Voici ce que vous devez obtenir :

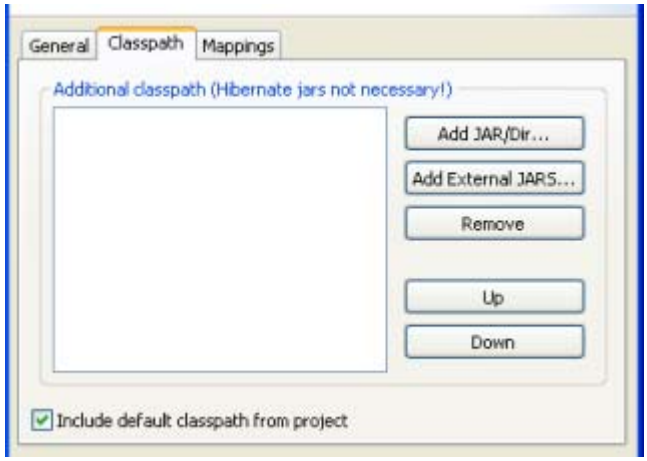


FIGURE 8. Gestion du `classpath`

Cliquer sur le bouton **`Finish`**, la configuration de console est créée.

3.2.1 Perspective HIBERNATE

Ouvrir la perspective **`Hibernate`** par le menu :

`Window>Open Perspective>Other>Hibernate`

Si la fenêtre **`Hibernate configuration`** n'est pas ouverte, ouvrez là par le menu **`Window>show view>Hibernate>Hibernate Configurations`**.

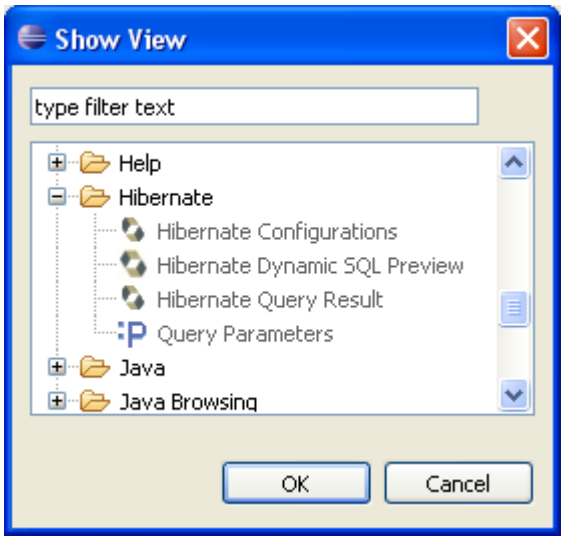


FIGURE 9. Ouvri la vue Hibernate Configurations

Vous devez voir apparaître la configuration que vous venez de créer. En déployant cette configuration vous devez voir apparaître les tables de la base **`film`** que vous avez créer au début de ce TP. Votre base a été accédée par HIBERNATE tools et sa structure vous est présentée :

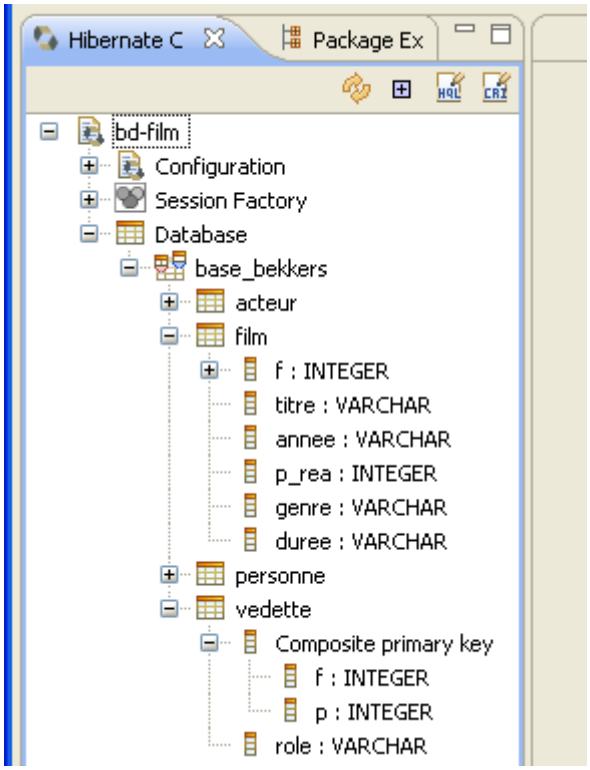


FIGURE 10. La base de données film vue par la configuration de console

3.3 Création du fichier d'ingénierie inverse reveng.xml

Pour adapter le processus d'ingénierie inverse à vos besoins et pour le contrôler, HIBERNATE utilise un document XML appelé **`reveng.xml`**. Vous devez donc préparer un tel fichier avant de lancer le processus de rétro-ingénierie. Le plugin **`Hibernate Tools`** fournit un assistant de création pour ce fichier.

Question 4. Génération du fichier reveng.xml

Lancez cet assistant par le menu :

`File>New>Other ...>Hibernate>Hibernate reverse Engineering File (reveng.xml)`

Dans la première fenêtre sélectionnez l'emplacement cible du fichier d'ingénierie inverse puis cliquez sur **`Next`**. *Dans la fenêtre suivante vous devez d'abord choisir une configuration de console, ici le choix est simple vous n'avez qu'une configuration, sélectionnez là puis cliquez sur* **`Refresh.`** *Le schéma de la base* **`film`** *doit alors apparaître dans la partie* **`Database schema`**. *Ensuite vous devez définir les tables que voulez prendre en compte :*

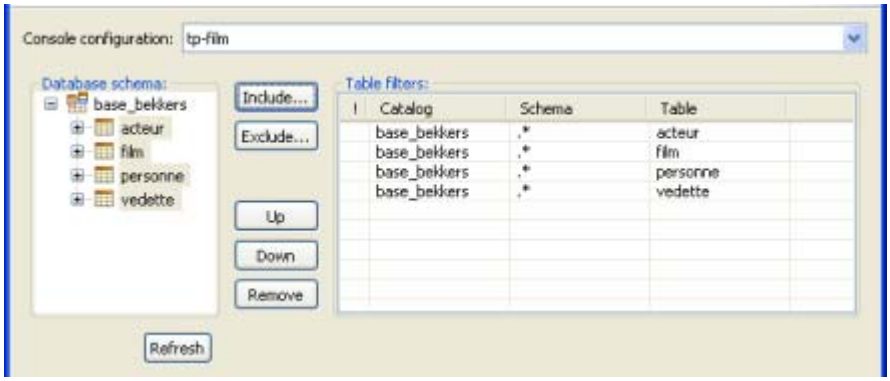


FIGURE 11. Inclure/exclure des tables

Ici, sélectionnez les quatre tables de la base **`film`** *et incluez les par le bouton* **`Include`**. *Terminer en cliquant sur le bouton* **`Finish`**.

Voici le document **`reveng.xml`** généré, il doit s'ouvrir automatiquement dans un éditeur spécifique :

```
<hibernate-reverse-engineering>
  <table-filter match-catalog="base_bekkers" match-name="acteur"/>
  <table-filter match-catalog="base_bekkers" match-name="film"/>
  <table-filter match-catalog="base_bekkers" match-
     name="personne"/>
  <table-filter match-catalog="base_bekkers" match-name="vedette"/
     >
</hibernate-reverse-engineering>
```

Cet éditeur est prévu pour faciliter la définition de la correspondance des types JAVA et de ceux de SQL. Il permet aussi d'inclure ou d'exclure des tables du processus d'ingénierie inverse. Enfin il permet de définir explicitement le nom

des champs si les règles de nommage par défaut d'HIBERNATE ne conviennent pas. Pour l'instant, vous n'avez pas à éditer ce fichier, les valeurs par défaut choisies par HIBERNATE nous conviennent.

3.4 Génération des classes JAVA correspondant au modèle de la base

Le plugin **`Hibernate Tools`** permet de lancer les diverses générateurs offerts par HIBERNATE. En particulier nous allons générer :

- Les classes JAVA du modèle,
- Les documents XML de «*mapping*» entre instances d'objet JAVA et lignes dans les tables

Sous Eclipse on lance les générateurs soit par le menu :

`Run>hibernate Code Generation...>hibernate Code Generation...`

soit par le bouton HIBERNATE  choisir

`hibernate Code Generation...`

La fenêtre qui s'ouvre permet de configurer la génération de code

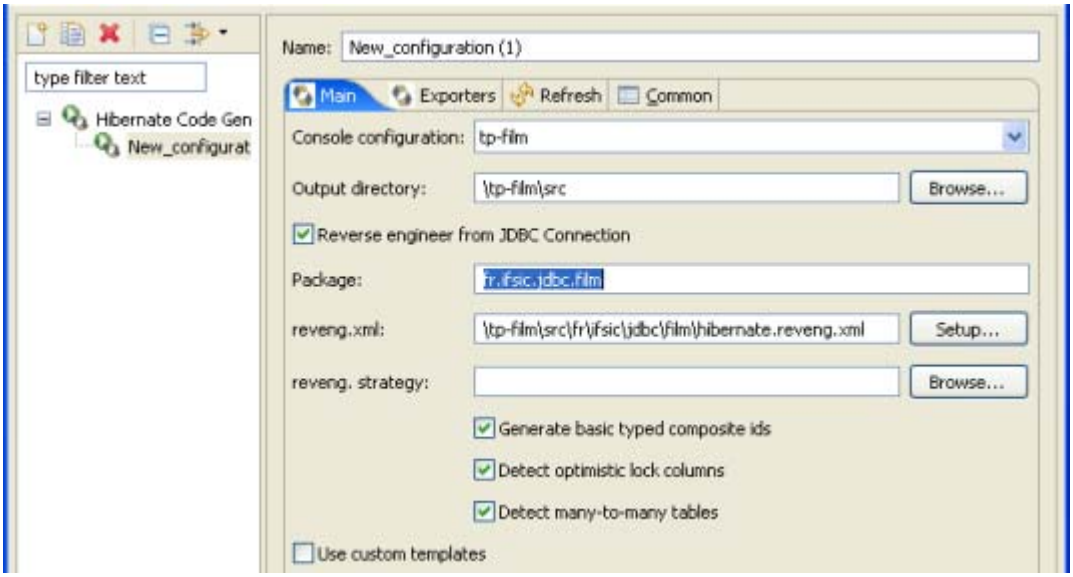


FIGURE 12. Configurer la génération de code

Vérifiez :

- que la configuration de console est la bonne.
- que le répertoire de sortie **`Output directory`** est votre répertoire des sources,
- que l'option **`Reverse engineering from JDBC Connection`** est bien sélectionnée. Vérifiez le nom de paquetage.

puis passez dans l'onglet **`Exporters`** :

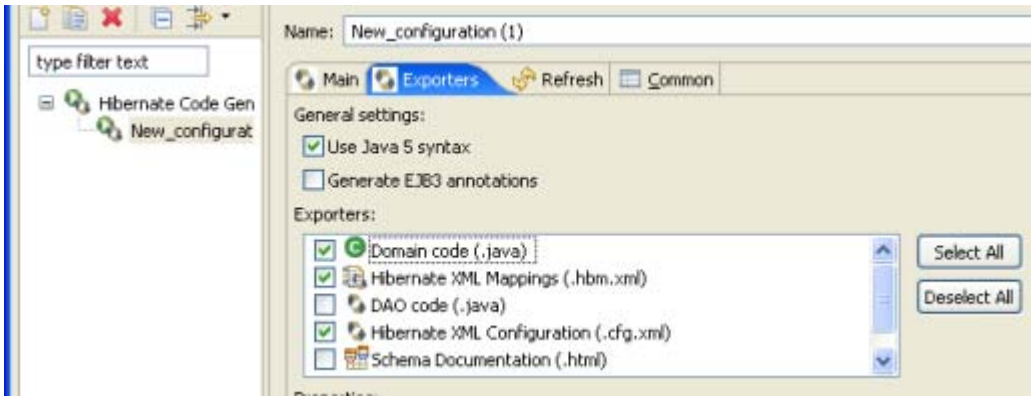


FIGURE 13. Choix des exporteurs

Choisissez les générateurs suivants :

1. Domaine Code
2. HIBERNATE XML Mapping
3. HIBERNATE XML Configuration

Cliquez sur **`Finish`** pour lancer la génération. Pour voir le résultat de cette génération revenez dans la perspective Java. Vous devez voir apparaître :

- les classes JAVA de votre modèle (ici les classes **`Film, Personne, Vedette`**...),
- des fichiers de «**`mapping.xml`**» objets/relation
- le fichier de configuration **`hibernate.cfg.xml`** que vous avez précédemment préparé doit avoir été modifié. Regardez le,. Que constatez vous ?

La vue package explorer de votre projet doit ressembler à ceci :

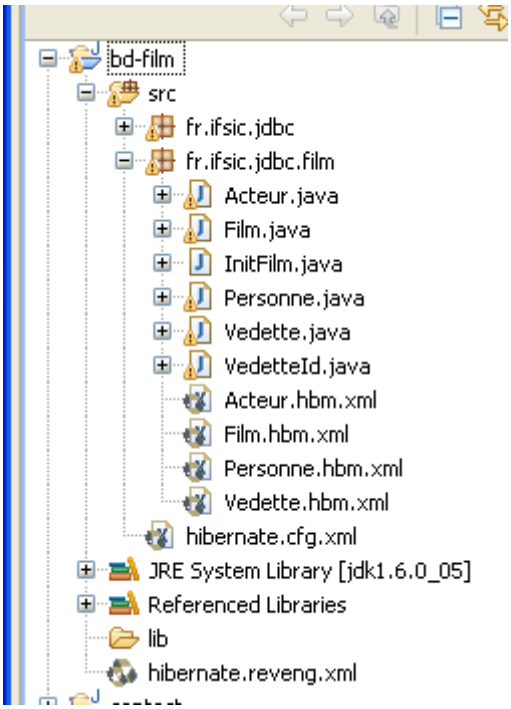


FIGURE 14. Vue package explorer après la génération de code.

Le processus d'ingénierie inverse est terminé, vous êtes prêt à accéder à la base de données en utilisant HIBERNATE.

3.4.1 Documents de mapping Objets Java/relations

Avant de continuer, parcourez les fichiers qui viennent d'être générés, en particulier observez les fichiers **`xxx.hbn.xml`**. Ces documents XML définissent le «*mapping*» entre objets JAVA et tables relationnelles. Vous devez avoir constaté qu'une référence à chacun de ces fichiers a été placée dans votre fichier de configuration.

Ouvrez le fichier **`Personne.hbn.xml`**. Voici son contenu :

```
<hibernate-mapping>
    <class name="fr.ifsic.jdbc.film.Personne" table="personne"
     catalog="base_bekkers">
        <comment></comment>
        <id name="p" type="int">
            <column name="p" />
            <generator class="assigned" />
        </id>
        <property name="nom" type="string">
            <column name="nom" length="12" not-null="true">
                <comment></comment>
            </column>
        </property>
        <property name="prenom" type="string">
            <column name="prenom" length="12">
                <comment></comment>
            </column>
```

```
        </property>
        <property name="sexe" type="string">
            <column name="sexe" length="1">
                <comment></comment>
            </column>
        </property>
    </class>
</hibernate-mapping>
```

Dans ce document XML on constate que le générateur d'HIBERNATE a reconnu que les quatre attributs de la table Personne. L'attribut «**p**» a été reconnu comme clés identifiant les éléments **`Personne`**. L'élément

```
<generator class="assigned" />
```

Signifie que HIBERNATE a décidé de laisser la génération des identifiants à l'utilisateur. Ceci est une tâche ingrate et sujette à erreur. HIBERNATE autorise, si l'utilisateur le désire, de s'en remettre pour cette tâche soit à HIBERNATE lui-même soit au SGBDR. MYSQL possède un mécanisme interne de génération d'identifiant, nous nous proposons de l'utiliser. Pour cela on vous demande de changer la valeur de l'attribut **`@class`** en «**`native`**».

```
<generator class="native" />
```

Cette modification sera utile plus tard dans le **`tp`**. Faites de même pour l'identifiant de la table **`Film`**.

4 Travailler avec les objets persistants

4.1 Création de session

Pour créer une session HIBERNATE on procède en deux étapes, création d'une fabrique de session, création d'une session :

```
SessionFactory sessionFactory;
try {
   // étape 1 : Crée la SessionFactory
   sessionFactory =
   new Configuration().configure().buildSessionFactory();
} catch (HibernateException ex) {
   throw new RuntimeException("Problème de configuration : "
   + ex.getMessage(), ex);
}
   // étape 2 : création d'une session
Session sess = sessionFactory.openSession();
```

On ferme un session par :

```
sess.close();
```

4.2 Classe HibernateUtil

Pour simplifier la tâche de gestion des sessions dans une application web dynamique, **`Hibernate`** fournit une classe **`HibernateUtil`** qui instancie une et

une seule fabrique pour toute l'application tout en fournissant des sessions HIBERNATE individuelles à chaque **`Thread`** qui la lui demande. Cette classe possède deux méthodes statiques :

```
public static Session currentSession()
public static void closeSession()
```

Utilisez ces deux méthodes pour gérer votre session HIBERNATE.

4.3 Lire un objet persistant

La méthode **`load()`** de **`Session`** vous donne un moyen de récupérer à partir de la base une instance d'objet persistant si vous connaissez son identification. Par exemple un obtient la personne d'identifiant «1» par :

```
Session sess = HibernateUtil.currentSession();
Personne p = (Personne) sess.load(Personne.class,new Integer(1));
```

La méthode **`load()`** ne retourne jamais de valeur **`null`**. Si l'objet est absent elle génère une exception. Utilisez cette méthode uniquement lorsque vous êtes certain que l'objet rechercher existe. Sinon utilisez la méthode **`get()`** qui a le même effet que **`load()`** mais qui retourne une valeur **`null`** si l'objet est absent. Avec **`get()`** l'utilisateur peut alors tester facilement la présence ou l'absence de son objet.

Question 5.

Surchargez la méthode **`toString()`** *de la classe* **`Personne`** *afin qu'elle génère une chaîne qui si on l'affiche dans la console présente une personne sous la forme :*

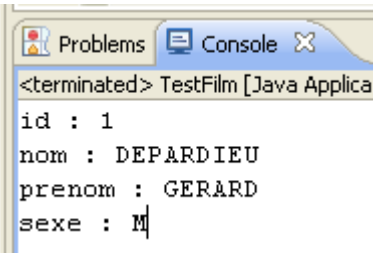


FIGURE 15. Interrogation de la table personne par un identifiant

Ecrire une méthode **`detailPersonne(sessions, id)`** *qui affiche la personne d'identifiant* **`id`** *en utilisant la session* **`session`**.

4.4 Rendre un objet persistant

Les instances nouvelles d'objet créées par new sont considérées comme éphémères par HIBERNATE. On peut rendre persistante une instance éphémère en l'associant à une session par la méthode **`save()`**

```
sess.beginTransaction();
Personne p = new Personne();
```

```
p.setNom("dupont");
...
Integer id = sess.save(p);
sess.getTransaction().commit();
```

Rappelons que nous avons associé précédemment le générateur «**native**» à l'identifiant «**p**» de la table **Personne**. C'est ce qui permet ici de laisser à la base de données le soin de générer l'identifiant. Ci-dessous voici le schéma du cycle de vie d'un objet dans HIBERNATE :

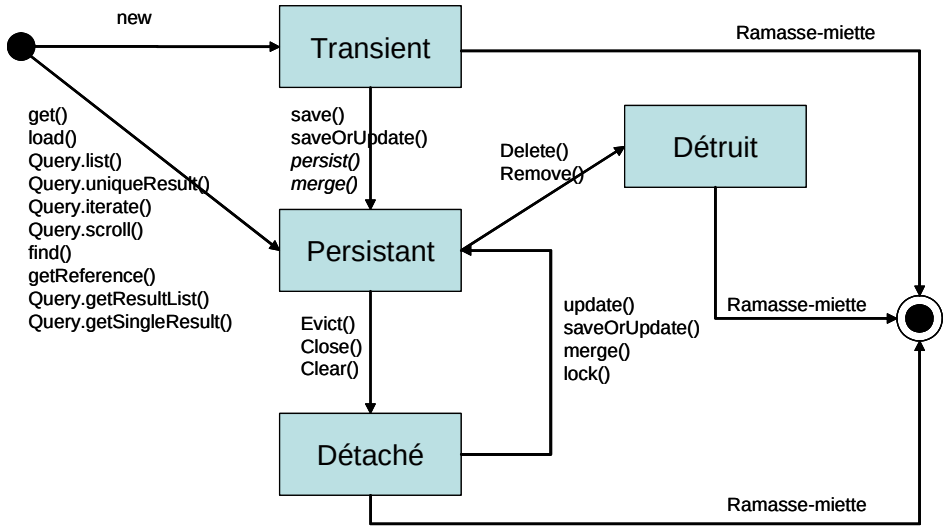


FIGURE 16. La vie des objets dans HIBERNATE

Question 6. Ajouter une personne

écrire une methode **`ajouter(session, p)`** *qui utilise la session* **`session`** *pour ajouter une personne nouvelle* **`p`** *à la base de donnée. Ne pas initialiser l'identifiant* **`p`** *de cette nouvelle personne. Vérifiez dans la base la présence de la nouvelle personne.*

Constatez qu'un identifiant unique a été généré par MYSQL. Lequel ?

Question 7. Effacer une personne

écrire une méthode **`effacer(session, p)`** *qui utilise la session* **`session`** *pour effacer une personne* **`p`** *de la base de données. Vérifiez dans la base que l'effacement a bien été effectué.*

Refléchissez au code qu'il aurait falut écrire en JDBC pour effectuer le même effacement.

4.5 Lancer une requête HQL ou SQL sur la base film

Le langage de requête HQL d'HIBERNATE permet d'effectuer des requêtes en gardant le niveau d'abstraction des objets JAVA définis par le mapping Objet/ Relation de votre application. Ainsi par exemple la requête HQL la plus simple ressemble à «**`from Personne`**», elle permet d'obtenir la liste de toutes les person-

nes se trouvant dans la base. Ici **Personne** est un nom de classe et non un nom de table. On pourrait demander de trier les personnes avec la requête HQL :

```
Query query = session.createQuery("from Personne p order by p.nom
    asc");
```

ou encore la requête SQL équivalente :

```
Query query = session.createSQLQuery("select p from Personne p
    order by p.nom asc");
```

On demanderait toutes les femmes en ordre croissant des noms par la requête SQL :

```
Query query = session.createSQLQuery("from Personne p where p.sexe
    = 'F' order by p.nom asc");
```

En résumé on prépare un requête HQL par la méthode **createQuery()** de l'objet **Session** et on prépare un requête SQL par la méthode **createSQLQuery()** de l'objet **Session**. Ces méthodes rendent une requête en résultat. On lance généralement une requête en appelant sa méthode **list()**. Cette methode rends en rsultat un tableau d'objets. Voici donc comment lancer une requête HQL par programme :

```
Session sess = HibernateUtil.currentSession();
sess.beginTransaction();
List<Object[]> result = List vehicule = sess.createQuery("from
    Vehicule v order by v.nom asc").list();
for (Object[] o : result) {
   Vehicule v = (Vehicule)o[0];
   System.out.println(""+v);
}
sess.getTransaction().commit();
```

La même requête peut être effectuées en SQL par la méthode **createSQLQuery()** de l'objet SESSION :

```
List<Object[]> result = sess.createSQLQuery("SELECT * FROM
    personne").list();
for (Object[] o : result) {
   System.out.println(""+o[0]+" "+o[1]+" "+o[2]+" "+o[3]);
}
```

ceci affiche pour chaque objet de la table la liste ses quatre attributs.

4.5.1 Tester une requête HQL avec `Hibernate tools`

Vous pouvez tester vos requêtes HQL sous ECLIPSE. Passez dans la perspective **Hibernate**. Dans votre configuration **tp-film** cliquer sur **session factory**. Ceci a pour effet de démarrer une session sur votre base de données. Ensuite un click droit dans la fenêtre **Hibernate Configurations** vous permet de démarrer un éditeur HQL. Dans l'entête de la fenêtre d'édition HQL sélectionnez votre configuration. puis tapez une requête **hql**.

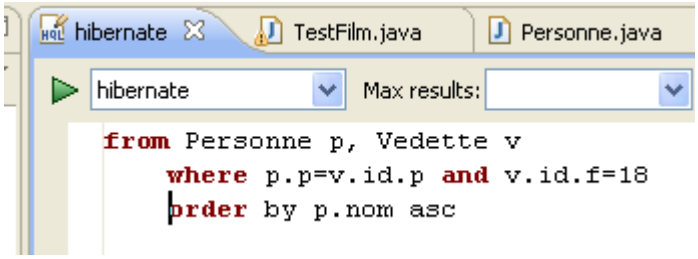


FIGURE 17. édition/exécution d'une requête `hql`.

Pour exécuter cette requête cliquer sur l'icône en forme de flèche verte. Les résultats s'affichent sous forme d'une liste d'objets dans la fenêtre **`Hibernate Query result`**. Ensuite il suffit de sélectionner un objet dans cette liste pour voir apparaître sa valeur dans la fenêtre **`Properties`** :

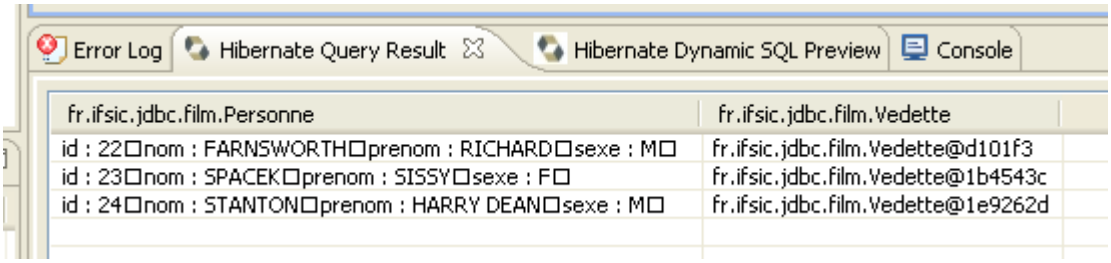


FIGURE 18. Visualisation des résultats d'une requête HQL

Enfin, il vous est possible de visualiser la requête SQL générée par votre requête HQL dans la fenêtre **`Hibernate Dynamic SQL preview`**.

Question 8.

écrire une méthode qui liste les acteurs d'un film donné par son nom.

5 Affinement du modèle de classes

5.1 Prise en compte de l'héritage

Dans le modèle de la base Film tout *acteur* est une *personne* mais toute personne n'est pas un acteur. Il y a une et une seule personne correspondant à un acteur et l'identifiant d'un acteur est le même que l'identifiant de la personne correspondante. Il s'agit là d'une relation «*un à un*» qui modélise l'*héritage*. La classe **`Acteur`** hérite de la classe **`Personne`**. Il y a plusieurs manières de représenter les relations d'héritage dans un modèle relationnel. La méthode qui a été employée ici est la méthode dite «*à une table par classe fille*». La classe **`Acteur`** est la classe **`Personne`** sont deux classes différentes. Une autre méthode parfois utilisée est celle «*à une table unique avec discriminant*» nous n'en parlerons pas ici.

Lors du processus d'ingénierie inverse à partir de la base, la relation d'héritage n'a pas pu être reconnue par HIBERNATE. La classe **`Acteur`** n'hérite pas de la classe **`Personne`**. Pourtant HIBERNATE permet la prise en compte de l'héritage pour peu que l'utilisateur le spécifie dans les documents de mapping. Nous allons donc revenir sur la génération de code, mais cette fois à partir des documents de mapping **`.hbn.xml`** et en spécifiant l'héritage présent dans ces documents. C'est ce que nous allons faire maintenant.

5.1.1 Modification des documents de mapping

Pour spécifier que la classe **`Acteur`** est une classes dérivée de la classe **`Personne`** procédez comme suit :

Question 9. Prise en compte de l'héritage

Effacez le document **`Acteur.hbn.xml`***, effacez sa référence dans le document* **`hibernate.cfg.xml`** *et effacez toutes les classes* JAVA *produites par* HIBERNATE *puis ajoutez en fin de document* **`Personne.hbn.xml`***, dans le contenu de l'élément* **`<class>`***, la définition* **`<joined-class>`** *suivante :*

```
<class name="fr.ifsic.jdbc.film.Personne" table="personne">
    <id name="p" type="int">
          <column name="p" />
          <generator class="native" />
    </id>
    ...
    <joined-subclass name="fr.ifsic.jdbc.film.Acteur"
        table="acteur" catalog="base_bekkers">
            <key column="p"/>
            <property name="nbFilm" type="string">
                <column name="nb_film" length="3"/>
            </property>
    </joined-subclass>
</class>
```

On spécifie ici que la classe **`Acteur`** *est une classe dérivée de la classe* **`Personne`** *avec une jointure explicitée par la clé commune* **`p`**.

Relancez les générateurs de code d'`Hibernate tools` *par le menu :*

`Run>hibernate Code Generation...>hibernate Code Generation...`

Vérifiez cette fois que la case **`Reverse engineer form JDBC Connection`** *n'est pas cochée :*

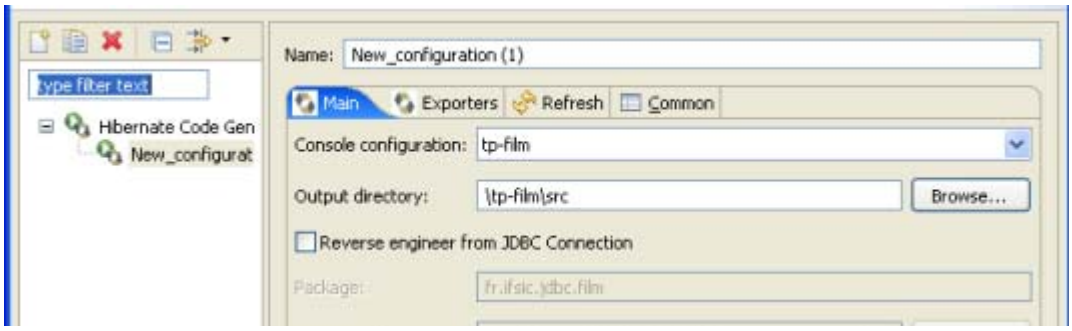
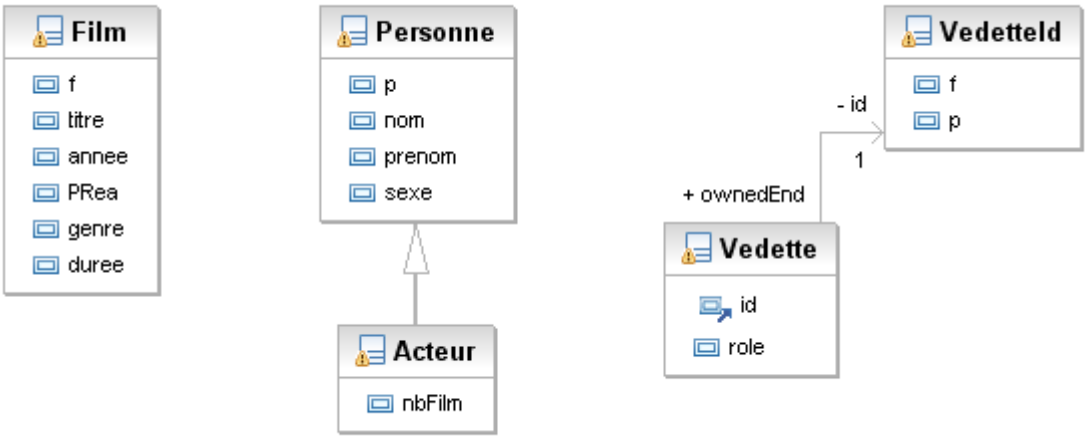


FIGURE 19. Génération des classes Java à partir des fichiers de mapping

Dans l'onglet **`Exporters`** *cochez seulement* **`Domain Code`** *cliquez sur* **`Finish`**.

Dans les sources Java vous pouvez vérifiez que la classe **`Acteur`** est bien maintenant une extension de la classe **`Personne`**. Voici ce que vous devez obtenir :



Question 10. Visualiser un acteur

En utilisant cet implémentation de l'héritage, écrire les méthodes **`toString()`** *qui visualisent les détails d'une personne et d'un acteur sachant que, lorsque la personne est un acteur, la méthode donne le nombre des films.*

```
P = 8
Nom : BINOCHE
Prénom : JULIETTE
Sexe = F
Nb film = 25
```

FIGURE 20. Affichage d'un acteur

5.2 Associations plusieurs à un

le modèle relationnel de la base **film** comporte trois associations *plusieurs à un* :

- l'attribut **PRea** de **Film**, désigne le réalisateur du film, un film possède un seul réalisateur et une personne peut être le réalisateur de plusieurs films.
- les attributs **f** et **p** de la table **vedette** désignent respectivement un film et une personne. Cette table est en fait une table de jointure entre les tables **personne** et **film**.

Ces trois associations n'ont pas été détectées par les générateurs de code d'HIBERNATE. On le voit dans le fichier de mapping **vedette.hbn.xml** :

```
<composite-id name="id" class="fr.ifsic.jdbc.film.VedetteId">
   <key-property name="f" type="int">
      <column name="f" />
   </key-property>
   <key-property name="p" type="int">
      <column name="p" />
   </key-property>
</composite-id>
```

Où les attributs **f** et **p** sont déclarés de type **int**. On le voit aussi dans le fichier de mapping **film.hbn.xml** où le type de l'attribut **PRea** est est lui aussi **int**. Vous pouvez vérifier que le type des attributs correspondants dans les classes **VedetteId** et **Film** est bien **int** alors que l'on aimerait que ce soit **Personne** ou **Film**.

Hibernate permet la prise en compte des relations plusieurs à un. Nous allons maintenant préciser notre modèle dans ce sens.

Question 11. Prise encompte des associations plusieurs à un

Remplacez dans le document **film.hbn.xml** *l'élément correspond à la propriété* **PRea** *:*

```
<property name="PRea" type="int">
   <column name="p_rea" not-null="true">
      <comment></comment>
   </column>
</property>
```

par :

```
<many-to-one name="PRea" column="p_rea"
   class="fr.ifsic.jdbc.film.Personne"/>
```

Puis remplacez la déclaration de l'identifiant de la table vedette :

```
<composite-id name="id" class="fr.ifsic.jdbc.film.VedetteId">
   <key-property name="f" type="int">
      <column name="f" />
```

```
   </key-property>
   <key-property name="p" type="int">
      <column name="p" />
   </key-property>
</composite-id>
```

par :

```
<composite-id name="id" class="fr.ifsic.jdbc.film.VedetteId">
   <key-many-to-one name="f" class="fr.ifsic.jdbc.film.Film"/>
   <key-many-to-one name="p" class="fr.ifsic.jdbc.film.Personne"/>
</composite-id>
```

Regénérez le code java des classes du domaine, sans repartir de la base, c'est-à-dire en partant de documents de mapping que vous venez de modifier. Vérifiez dans le classe **`VedettedId`** *le type des deux attributs qui doit correspondre maintenant aux types respectifs* **`Film`** *et* **`Personne`***. Quel est le type de l'attribut* **`prea`** *de la classe* **`Film`***.*

Question 12.

Selon vous que signifie la requête HQL correcte suivante

```
select distinct v.id.p.nom from Vedette v
```

Testez la. Cette requête aurait-elle été correcte avant les modifications que vous venez de faire ? Comment écririez vous cette requête en SQL ?

Question 13.

écrivez une fonction **`detailFilm()`** *qui rend une chaîne qui contient les informations sur un film donné y compris le nom et le prénom de son réalisateur.*

Remarquez que n'y pas besoin de faire une requête de jointure, une simple requête sur la table **`film`** *suffit, pourquoi ? Expliquez comment ça marche.*

Voici ce que vous devez obtenir :

Titre	HANNAH ET SES SOEURS
Genre	COMEDIE PSYCHO.
Année	1986
Duree	100 mn
Réalisateur	WOODY ALLEN
Id	3

FIGURE 21. Informations sur un film