UQAC

Travail pratique #3 : Conception du serveur d'application et du client

Cours	8TRD157 – Base de données avancées
Étudiant 1	Maël Bervet
Étudiant 2	Valentin Maurice

Rapport	**(45 pts)**
Manuel utilisateur	**/ 5**
Stratégie de test	**/ 5**
Patrons de conception	**/ 5**
Validation des contraintes et de l'accès	**/ 5**
Code de mappage et justification	**/ 10**
Question 1	**/ 5**
Question 2	**/ 5**
Autres items	**/ 5**
Code source	**(30 pts)**
Code Hibernate	**/ 10**
Code d'application Java	**/ 20**
Qualité du code	
Fonctionnalité	**(25 pts)**
Correction interactive	**/ 25**
Total	**/ 100**

Sommaire

1 Introduction

Après avoir modélisé une base de données pour un magasin de location de film en UML et l'avoir retranscrite en scripts SQL, à présent nous avons à programmer l'application associée en Java. Celle-ci est entièrement programmée sous Eclipse en suivant le modèle MVC. Différentes technologies sont utilisées afin de fournir une application aussi complète que viable. Le model est lié à la base de données par la technologie hibernate mapping qui permet de dissocier les requêtes du SGBD utilisé, et la vue est entièrement développée en utilisant javafx le créateur d'interface utilisateur d'Eclipse.

2 Manuel utilisateur

Prérequis :

Avant de lancer l'application, il est nécessaire d'avoir créé la base de données dans sa totalité (tables, triggers, procédures) et de lancer le script data.sql dans l'interface de commande oracle afin que l'application est les données nécessaires pour faire fonctionner les cas d'utilisations demandés (tous les scripts sont dans le dossier SQLScripts du projet).

Connexion d'un utilisateur :

La connexion est la première étape de l'utilisation de l'application. Tout d'abord vous avez trois choix soit vous vous connectez en tant que client, soit en tant qu'employé, soit vous abonner afin de créer un compte client. Si vous vous connectez en tant qu'employé, le compte administrateur est disponible avec l'identifiant "0" et le mot de passe "admin". Cependant pour les tests des clients ont été créé, vous pouvez donc vous connecter avec le courriel "Courriel3" et le mot de passe "password" ce qui vous emmènera sur la page de consultation des films.

Consultation interactive des films :

La consultation des films se fait via à l'onglet film, en utilisant les différents critères de recherche mis à disposition. Les correspondances retournées sont ensuite affichées.

Location d'un film :

Une fois que le film recherché a été trouvé vous pouvez louer celui-ci simplement en cliquant sur le bouton "Louer" associé à ce film, si toutefois votre forfait vous le permet.

3 Stratégie de test

Notre stratégie de test est assez simple, lorsqu'une fonctionnalité est programmée celle-ci est testée par rapport au différents cas d'utilisation et contraintes qui lui sont associés et cela grâce à un script SQL remplissant la base de données avec des données fictives.

4 Patrons de conception employés

Utilisant le mapping avec hibernate dans un logiciel développé en java sous Eclipse aucun patron de façade, courtier BD, DAO ou DTO n'a été utilisé.

5 Validation des contraintes

Toutes nos règles d'affaire ou contraintes ont été gérées dans la base de données ou dans l'application de la manière suivante :

Contraintes inhérentes au Model:

Table(s)	Contrainte	Stratégie
Employes	Un employé peut exister uniquement si la personne existe	Les champs "Nom" et "Prenom" Sont mis en FOREIGN KEY de la table "Personnes" et NOT NULL
Clients	Un client peut exister uniquement si la personne existe	Les champs "Nom" et "Prenom" Sont mit en FOREIGN KEY de la table "Personnes" et NOT NULL
Clients	Le client doit obligatoirement avoir un forfait existant	Le champs "TypeForfait" est mis en FOREIGN KEY de la table "Forfaits" et NOT NULL
Films, Personnes, Scenaristes, RolesActeurs	Les correspondances films personnes scénariste et acteurs doivent être respectées	Les quatres tables sont liées grâce aux FOREIGN KEY "Nom", "Prenom" et "TitreFilm" qui sont mis à NOT NULL
Forfait	Un forfait doit obligatoirement avoir ses caractéristiques afin d'éviter les erreurs	Application du critère NOT NULL sur tous les champs

Cas d'utilisation 1:

Table(s)	Contrainte	Stratégie
Clients	Aucun champs vide	Tous les champs sont NOT NULL
Clients	Il ne peut y avoir deux mêmes courriels	Champs "Courriel" UNIQUE
Clients	Carte visa, mastercard et amex	Application d'un CHECK IN sur le champs "TypeCarte"
Clients	La carte de crédit ne doit pas être expirée	Utilisation d'un TRIGGER sur l'insertion d'un date d'expiration
Clients	Le client doit avoir au moins 18 ans	Utilisation d'un TRIGGER sur l'insertion d'une date de naissance
Clients	Minimum 5 caractères alphanumériques pour le mot de passe	Utilisation d'un check sur le champs "MotDePasse"
Forfaits	Respect du nombre de locations	Vérification lors de la demande dans le controler
Forfaits	Respect de la durée de location	Vérification dans le controler

Cas d'utilisation 3:

L'unique contrainte identifiée dans ce cas d'utilisation est que tous les champs des tables "Films", "Personnes", "Scenaristes" et "RolesActeurs" doivent être remplis. On les a donc tous mis NOT NULL.

Cas d'utilisation 7:

Table(s)	Contrainte	Stratégie
Films	La fiche du film doit être remplie	Tous les champs NOT NULL
Films	Un nouveau film doit avoir au moins une copie	Vérification du nombre de copies à la demande de création
Films	Un film ne peut être supprimé uniquement si toute les copies sont retournées	Utilisation d'un TRIGGER lors de la suppression d'un film

Cas d'utilisation 8:

Table(s)	Contrainte	Stratégie
Clients	Un client ne peut être supprimé tant qu'il n'a pas retourné tous ses films	Utilisation d'un TRIGGER lors de la suppression d'un client
Forfaits	Respect de la durée de location	Vérification dans le controler du client avec affichage de liste

6 Validation de l'accès et sécurité du système

Les sécurités mises en place sont, tout d'abord une connexion obligatoire, soit en tant qu'employé soit en tant que client, afin que l'application ne puisse être utilisée que si l'utilisateur s'est identifié. Ensuite, en fonction du type d'utilisateur celui-ci aura accès à plus ou moins de fonctionnalités. Enfin nous voulions que tous les mots de passe soit hachés avant enregistrement, afin qu'il ne soit pas visibles aussi bien sur le réseau que dans la base de données, cependant il ne fut pas possible de le mettre en place par manque de temps.

7 Modifications au schéma

L'unique modification apportée au schéma et l'ajout d'un champs "DateLocation" dans la table "Copies".

8 Difficultés rencontrées avec Hibernate

Les deux difficultées principales que nous avons rencontré sont le mappage de deux de nos classes associatives (*Scenariste, RolesActeur*) car leurs clés primaires sont la combinaison de trois champs qui sont également des clés étrangères et la compréhension de la correspondance des tables en Java avec utilisation des "Set" qui a entraîné des difficultés conséquentes pour la programmation des requêtes.

9 Code de mappages Hibernate pour les films, acteurs et réalisateurs

Mapping film :

```
<class dynamic-insert="false" dynamic-update="false" mutable="true" name="Model.Films" optimistic-lock="version" polymorphism="implicit" schema="TPBDD" select-before-update="false" table="FILMS">

  <id name="titre" type="string">
    <column length="100" name="TITRE"/>
    <generator class="assigned"/>
  </id>

    <property generated="never" lazy="false" name="annee" optimistic-lock="true" type="int" unique="false">
    <column name="ANNEE" not-null="true" precision="6" scale="0"/>
  </property>

    <property generated="never" lazy="false" name="duree" optimistic-lock="true" type="short" unique="false">
    <column name="DUREE" not-null="true" precision="4" scale="0"/>
  </property>

    <property generated="never" lazy="false" name="langue" optimistic-lock="true" type="string" unique="false">
    <column length="20" name="LANGUE" not-null="true"/>
  </property>

    <property generated="never" lazy="false" name="resume" optimistic-lock="true" type="string" unique="false">
    <column length="1000" name="RESUME" not-null="true"/>
  </property>

    <property generated="never" lazy="false" name="genres" optimistic-lock="true" type="string" unique="false">
    <column length="200" name="GENRES" not-null="true"/>
  </property>

    <property generated="never" lazy="false" name="paysproduction" optimistic-lock="true" type="string" unique="false">
    <column length="200" name="PAYSPRODUCTION" not-null="true"/>
  </property>
```

```
        <set embed-xml="true" fetch="select" inverse="true" lazy="false" mutable="true" name="rolesacteurses" optimistic-lock="true" sort="unsorted" table="ROLESACTEURS">
      <key on-delete="noaction">
        <column length="100" name="TITREFILM" not-null="true"/>
      </key>
      <one-to-many class="Model.Rolesacteurs" embed-xml="true" not-found="exception"/>
    </set>

    <set embed-xml="true" fetch="select" inverse="true" lazy="false" mutable="true" name="scenaristes" optimistic-lock="true" sort="unsorted" table="SCENARISTES">
      <key on-delete="noaction">
        <column length="100" name="TITREFILM" not-null="true"/>
      </key>
      <one-to-many class="Model.Scenaristes" embed-xml="true" not-found="exception"/>
    </set>

     <set embed-xml="true" fetch="select" inverse="true" lazy="false" mutable="true" name="copieses" optimistic-lock="true" sort="unsorted" table="COPIES">
      <key on-delete="noaction">
        <column length="100" name="TITREFILM" not-null="true" unique="true"/>
      </key>
      <one-to-many class="Model.Copies" embed-xml="true" not-found="exception"/>
    </set>

</class>
```

Mapping acteur :

```
<class dynamic-insert="false" dynamic-update="false" mutable="true" name="Model.Rolesacteurs" optimistic-lock="version" polymorphism="implicit" schema="TPBDD" select-before-update="false" table="ROLESACTEURS">

  <composite-id class="Model.RolesacteursId" mapped="false" name="id" unsaved-value="undefined">
    <key-many-to-one class="Model.Personnes" name="personnes">
      <column length="30" name="NOM"/>
      <column length="30" name="PRENOM"/>
    </key-many-to-one>
    <key-many-to-one class="Model.Films" name="films">
      <column length="100" name="TITREFILM"/>
    </key-many-to-one>
  </composite-id>

      <property generated="never" lazy="false" name="nompersonnage" optimistic-lock="true" type="string" unique="false">
    <column length="40" name="NOMPERSONNAGE" not-null="true"/>
  </property>

</class>
```

Mapping scénariste :

```
<class dynamic-insert="false" dynamic-update="false" mutable="true" name="Model.Scenaristes" optimistic-lock="version" polymorphism="implicit" schema="TPBDD" select-before-update="false" table="SCENARISTES">

  <composite-id class="Model.ScenaristesId" mapped="false" name="id" unsaved-value="undefined">
    <key-many-to-one class="Model.Personnes" name="personnes">
      <column length="30" name="NOM"/>
      <column length="30" name="PRENOM"/>
    </key-many-to-one>
    <key-many-to-one class="Model.Films" name="films">
      <column length="100" name="TITREFILM"/>
    </key-many-to-one>
  </composite-id>

</class>
```

10 Planification des tâches

La répartition des tâches s'est, dans un premier temps, faite de manière assez simple, le premier arrivé avançait le projet. Une fois le projet, le GIT créé et la partie model codée, les tâches ont été réparties à l'avance de manière équitable considérant que l'un des protagonistes avait formulé le souhait que l'application soit codée dans sa totalitée et que le second avait accepté.

D'où ce tableau de répartition des tâches :

Tâches	**Protagoniste**
Création et structuration du projet	Maël Bervet
Mise en place du GITHUB	Valentin Maurice
Programmation de la partie model	Maël Bervet
Mapping des classes à la BDD	Tous
- Films, Personnes, Copies, RolesActeurs, Scenaristes	Maël Bervet
- Utilisateurs, Clients, Employes, Forfaits	Valentin Maurice
Programmation de la partie controler	Maël Bervet
Programmation de la partie vue	Tous
- cas d'utilisation 2, 3, 4 et 5	Valentin Maurice
- cas d'utilisation 1, 6, 7 et 8	Maël Bervet

Enfin voici le tableau de répartition des tâches par rapport au temps passé sur le travail demandé :

Tâches	**Maël Bervet**	**Valentin Maurice**
Modélisation des classes de l'application en java	90%	10%
Mappage des objets à la BDD avec hibernate	30%	70%
Implémentation de la couche présentation	0%	100%
Rédaction du rapport	95%	5%

11 Question théorique 1

L'utilisation d'un framework ORM tel que Hibernate permet de créer une application, logiciel ou web, qui n'est pas dépendante du SGBD utilisé pour gérer les données dont la persistance est gérée de manière transparente. Cela offre à l'application une plus grande adaptabilité. De plus ne pas avoir à se soucier de la syntaxe des requêtes permet de faciliter très largement la maintenabilité de cette application ainsi que les différents tests. Enfin l'utilisation d'un tel outil permet une meilleure gestion de la mémoire cache et diminue le nombre d'accès à la base de données.

12 Question théorique 2

Un patron de façade dans une application à plusieurs couches permet d'ajouter de la transparence entre les différentes couches de l'application les rendants ainsi plus indépendantes les unes des autres. Cela simplifie l'utilisation d'un module depuis un autre en plus de faciliter de manière considérable la maintenabilité de l'application. Cela s'apparente à ce que fait le controleur pour la vue dans un modèle MVC. Celui-ci permet la manipulation et l'utilisation du modèle de manière plus simple et transparente pour la vue.

13 Conclusion sur le projet

Afin de conclure sur le projet dont le principal but était d'apprendre à gérer les données de manière plus transparente en utilisant Hibernate. On peut dire que l'utilisation de ce framework simplifie énormément le code de la couche model de l'application et rend celui-ci également plus clair, lisible et maintenable. Malheureusement, malgré notre volonté de coder cette application dans sa totalité le manque de temps nous a rattrapé, ce qui nous a empêché de finir la couche vue de l'application et également de mettre en place des classes de test unitaires qui auraient permis de performer la robustesse des fonctions et de grandement faciliter les tests des différentes parties de l'application. Cependant, dans un même lapse de temps, dans une entreprise à travailler chaque jour sur une application tel que celle-ci, la programmer entièrement aurait été tout à fait possible.