**Base de données objet**

**DIFFERENCE ENTRE BD RELATIONELLE ET BD OBJET**

Modèle relationnel

•Basé sur les données

•Montre les relations entre les données

•Basé sur un modèle mathématique formel

•Le lien vers des programmes objets est plus difficile

Modèle objet

•Représente les objets du système et leur structure

•Permet de voir les associations, les compositions, les agrégations et l’héritage entre les objets

•Peut être utilisé pour implémenter directement une solution

DESAVANTAGE DU MODELE DE DB OBJECT

Avec un modèle relationnel, le SGBD s’occupe de s’assurer que les contraintes du modèle sont respectées

•Avec une BD objet, il faut faire le traitement manuellement

•Ajoute une complexité au code!

Gestion manuelle des contraintes d’intégrités entre les objets

**Persistance :**

L’objet est créé durant l’exécution d’un programme et sauvegardé à la fin de l’exécution

Transcient : L’objet n’est pas persistant

DIFFERENCE ENTRE BD OBJET ET LANGAGE OBJET

Dans les langages de programmation une classe est seulement un type

Dans un système de gestion de base de données objet, une hiérarchie de classe dénote une hiérarchie de type Une classe est un **type** et l’**ensemble** des instances de ce type.

**ObjectDB**

ObjectDB est un système de gestion de base de données objet pour Java

JPA est souvent utilisé dans le développement de solution ORM, contrairement à JDO

ObjectDB utilise le concept d’entité pour sauvegarder les instances des classes.

•Pour qu’une classe puisse être rendue persistante, elle doit spécifiée l’annotation Entity de JPA

Une application doit pouvoir se connecter sur la base de données objet afin de rendre persistant les objets •La connexion se fait à l’aide des classes EntityManagerFactory et EntityManager

•Le EntityManagerFactory permet de faire la connexion

•Le EntityManager permet de gérer les objets

Le commit s’assure de mettre à jour la base de données

Version orienté objet de SQL

JPA, et ObjectDB, offre un mécanisme pour effectuer des requêtes sur les objets d’une base de données

**JPQL - Requête**

TypedQuery<Country> q1 = em.createQuery

("SELECT c FROM Country c", Country.class);

List<Country> results = query.getResultList();

Le servlet va

•Extraire les paramètres de la requête

•Valider les types des paramètres

•Appeler la méthode appropriée du gestionnaire de transactions

**Servlet**

•Pour le servlet, il n’y a pas de différence entre la méthode doPost et la méthode doGet

•Pour le client

•La méthode GET transmet la requête en un seul bloc au serveur

•Le nom du servlet

•Les paramètres du formulaire

•La méthode POST transmet la requête en deux blocs

•Le nom d’un servlet en premier

•Les paramètres du formulaire en deuxième

titre

Acteur

serie

episode

nom

datenaissance

lieunaissance

anneeDebut

anneeFin

description

nbsaison

titre

nbsaison

nepisode

dureé

DateDiffusion

description

Distribution

Fait partie

n

1

n

n

role

1. Écrivez la déclaration des classes de la base de données en utilisant la notation ODL. Vous n’avez pas donner les méthodes des classes. Considérez que les classes Passagers et Aéroport existent déjà et peuvent être utilisés.

Class Acteur (extent allActeur, key idActeur) {

attribute int idActeur;

attribute datenaissance;

attribute lieunaissance;

relationship set<Episode> episode;

inverse Episode ::acteur;

}

Class Episode (extent allEpisode, key idActeur) {

attribute int idEpisode;

attribute nEpisode

attribute nbsaison

attribute titre

attribute duree

attribute dateDiffusion

attribute description

relationship set<Acteur> acteur;

inverse Acteur :: episodes;

relationship Serie serie;

inverse set Serie :: episode;

}

Class DistributionActeur (extent allDistribution, key (idActeur, idEpisode)){

attribute idActeur

attribute idEpisode

attribute role

relationship set<Acteur> acteur;

}

Class Serie(extent allSerie, key dateDiffusion ){

attribute dateDiffusion;

attribute titre

attribute anneeFin

attribute description

attribute nbsaison

relationship set<Episode> ep;

inverse Episode::acteurs

}

b. Écrivez les commandes OQL qui permettent d’effectuer les requetes suivantes en vous basant sur les classes definies en a) :

- Obtenir la liste de tous les acteurs qui ont joués dans l’épisode Dooms day

SELECT a

FROM a IN allActeur

WHERE a.episode.titre= “Dooms day”

* Obtenir la liste de tous les acteurs qui ont joués dans la serie 24h chrono

SELECT a

FROM a IN allActeur

WHERE a.episode.serie.titre = “24h chrono”

* Obtenir le nom et la date de naissance des acteurs qui ont joués dans la serie 24 chrono

SELECT struct (nom : a.nom,

Date de naissance : a.datenaissance)

FROM a in allActeur WHERE a.episode.serie.titre = “24h chrono”;

Ou

SELECT struct (nom : a.nom,

Date de naissance : a.datenaissance)

Serie : SELECT s.titre

IN t allSerie

WHERE s.titre = “24h chrono”

FROM a in allActeur

C. Écrivez le code de définition de la classe Episode en java en utilisant ObjectDB. Vous n’avez pas à faire les méthodes, ni les constructeurs. Votre réponse doit contenir la déclaration de la classe et des variables seulement.

@Entity

Public class Episode

{

@Id

@GeneratedValue

private long m\_id;

private int nepisode

private int nsaison

private string Titre

private long Duree

private Date dateDiffusion

private string description

@ManyToOne

Private Serie m\_serie;

Question 2 MONGO DB

Donnez le code java pour le gestionnaire de collections serie

* La méthode AjoutEpisodeDansSerie

Public AjoutEpisodeDansSerie(TupleEpisode e, Date anneeDebut)

{

serieColl.updateOne(eq (“anneeDebut”, anneeDebut)),

push (“Episode”, e.toDocument)

}

Public class TupleEpisode

{

private m\_nepisode ;

private m\_nbsaison ;

private m\_titre ;

private m\_duree ;

private m\_dateDiffusion ;

private m\_description ;

Public Document toDocument()

{

Return new Document().append(“nepisode”,m\_ nepisode)

.append(“nbsaison”,m\_nbsaison)

.append(“titre”,m\_titre)

.append(“duree”,m\_duree)

.append(“dateDiffusion”,m\_dateDiffusion)

.append(“description”,m\_description)

}

}

* La méthode AjoutSerie, cette methode ajoute une serie dans une base de donnneés

public void AjoutSerie(TupleSerie s)

{

serieColl.insertOne(s.toDocuement)

}

Public class TupleSerie

{

……….

return new Document().append(“anneeDebut”,m\_ anneeDebut)

.append(“anneeFin”,m\_ anneeFin)

.append(“titre”,m\_titre)

.append(“nbsaison”,m\_ nbsaison)

.append(“description”,m\_description)

}

}

* La méthode getActeur(int idActeur). Cette méthode retourne l’acteur associé au numéro d’identification spécifié en paramètre

public TupleActeur getActeur(int idActeur) {

Document d = acteursColl.find(eq("idActeur ", idActeur)).first()

If( d!=null) {

return new TupleActeur(d);

}

return null

}

public class TupleActeur

{

………..

public TupleActeur(Document d)

{

m\_idActeur =d.getInteger("idActeur");

m\_datenaissance=d.getDate("datenaissance");

m\_lieunaissance=d.getString("lieunaissance")

}

}

Question 3

Codez un servlet et une page JSP qui permet d’afficher tous les acteurs sur une épisode particulière. Pour calculer la liste des acteurs, votre servlet ou votre page JSP doit appeler la méthode suivante :

List<Acteur>getActeur (String idActeur)

Réponse :

Public void doPost(HttpServletRedquest request, HttpServletResponse response) {

try {

String idEpisode = request.getSession().getAttribute(id)

If (idEpisode == null || idEpisode == " ")

Throw new EException ( "numéro de l’épisode invalide ")

Episode e = (Episode)getServletContext().getAttribute("episode ");

List<Acteurs>a;

Synchronized(e) {

a = e.getActeurs(idEpisode)

}

request.setAttribute(" Acteurs ", a)

requestDispatcher dispatcher = resquest.getRequestDispatcher("/WEB-INF/"listeActeurs.jps);

respond.sendRedirect(" WEB-INF/erreur.jps")

}

Catch{

Response.sendRedirect("WEB-INF/erreur.jps")

}

}

* Ne donnez que le code de la méthode doPost du servlet et seulement la partie de la

page JSP qui affiche la liste des noms des passagers, selon les en-têtes suivants :

<table><tbody>

<tr><td>Acteur</td></tr>

... votre code JSP irait ici ...

</tbody></table>

**solution**

<table><tbody>

<tr><td> Acteur </td></tr>

<% < List <Acteur> acteurs = (List<Acteur>)request.getAttribute("acteurs");

for(Acteur a : acteurs)}

%>

<tr>

<td><%=a.getNom()%></td>

<td><%=c.getDateNaissance()%></td>

<td><%=c.getLieuNaissance()%></td>

</tr>

<%

}

%>

</tbody>

</table>