Département de génie logiciel et des T.I.

RAPPORT DE LABORATOIRE

Numéro du laboratoire	**01**
Étudiant(s) et code permanent	Alexandre Laroche LARA12078907 Carl-Henri Codio CODC11157808 Esdras Metayer METE21058607 DIALLO Amadou Oury DIAA26018801 Sana Bouhadida BOUS27518509
Numéro d'équipe	**06**
Cours	**GTI660**
Groupe	**01**
Chargés(es) de laboratoire	**Mirna Awad**
Date	11 février 2019

TABLE DES MATIÈRES

1. INTRODUCTION

Ce présent laboratoire consiste à développer un système de base de données de location de films. Il est une introduction à l'architecture de bases de données multimédias qui permet de se familiariser aux notions de base des requêtes SQL telles que la création, l'interrogation, la modification des objets SQL ainsi que les importations de données existantes.

Une analyse minutieuse des données disponibles dans les fichiers personnes, clients et films sera faites afin de respecter les exigences décrites dans l'énoncé qui sont entre autres la conception d'un modèle conceptuel et d'un schéma relationnel de données, l'implémentation de script pour la création de la structure des tables, l'ajout de contraintes sur les tables, la validation des contraintes et l'importation des données du fichier xml vers la base de données.

Les prochaines lignes présentent la conception dans son ensemble en commençant par l'analyse du fichier xml qui contient les données nécessaires pour la création des classes de notre base des données. La conception du schéma relationnel qui traduit le modèle conceptuel en transformant nos classes en tables. Puis, on abordera l'implémentation de l'algorithme utilisé pour réaliser le programme permettant l'insertion des données et les différents moyens utilisés pour la validation des contraintes, une courte discussion et on finira par une conclusion.

2. ANALYSE

D’une part, le système à développer doit répondre aux requêtes de l’utilisateur afin de pouvoir interroger, modifier et d’interagir avec la base de données. D’autre part, la conception de la base de données doit prendre en considération la notion de contraintes présentées dans le cours afin d’assurer la cohérence et l’intégrité des données.

Pour ce faire, un modèle conceptuel est conçu dans le but de présenter l’ensemble des classes, leurs attributs et les différentes relations reliant les classes entre eux. Par exemple les classes ***Rôle***, ***Client*** et ***Réalisateur, Scénariste*** qui héritent de la classe ***Personnes*** puisqu'elles ont toutes des attributs communs tels que: idpersonne, nom, prénom, dateDeNaissance et autres. Pour les classes qui présentent des relations de multiplicité plusieurs à plusieurs (N à N), telle que ***Genre et Films***, ***Pays et Films*** la conversion vers le modèle relationnel transforme les relations en classe comme indiquées dans le schéma relationnel ci-dessous. Les autres entités comme ***Annonce et infocrédit*** sont directement liées à une autre entité via une relation de un à plusieurs ou bien de un à un selon le cas. Ainsi, un film est tourné par un et un seul Scénariste et le film peut être tournés par un à plusieurs scénaristes, un film ne peut appartenir qu’à une seule catégorie et une catégorie peut avoir un à plusieurs films. Une Personne peut avoir un à plusieurs rôles dans un film.

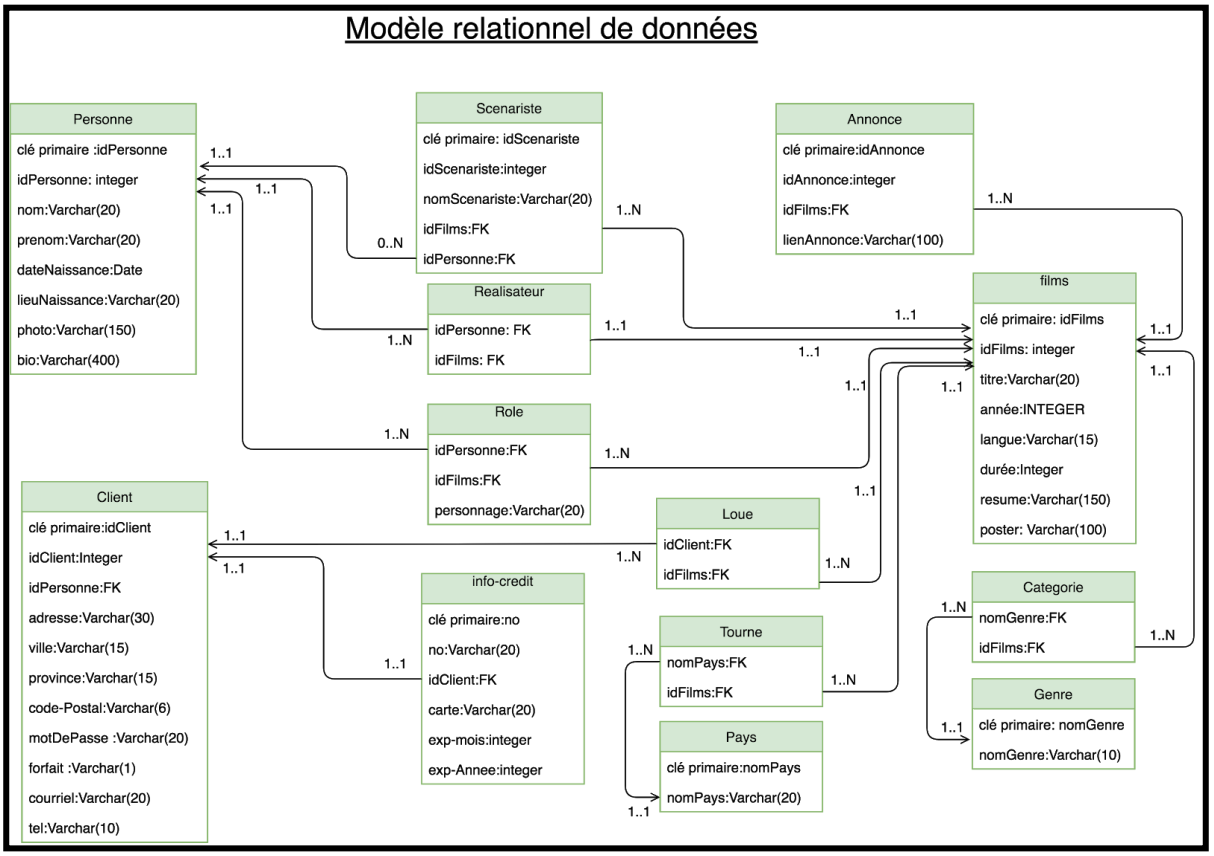
La conception du modèle relationnel par contre a permis d’identifier plusieurs contraintes telles que: les contraintes d’intégrité, les contraintes de domaine et les contraintes d’intégrités référentielles. Elles sont définies par plusieurs éléments dans une table comme les PRIMARY KEY, FOREIGN KEY, NOT NULL, CHECK et autres. Dans la table Client par exemple, le IDCLIENT qui est défini comme étant la clef primaire afin maintenir l’unicité des données va se voir imposer la contrainte NOT NULL puis qu’il ne peut pas être vide.

3. CONCEPTION

Le modèle relationnel de notre base de données a été structuré à partir du modèle conceptuel. En effet, les classes ont été transformées en tables. Pour chaque table, nous avons identifié les attributs et la clé primaire et/ou étrangère la caractérisant.

Le type de chaque attribut a été transformé en un type Oracle. Nous avons par la suite procédé à la détection des différentes contraintes d'intégrité régissant les différentes tables.

Schéma relationnel de données



La table ***personne*** du modèle ne servira à persister les informations sur les différents types d'intervenants dans le système. Ainsi les tables ***rôles*** et ***réalisateurs*** n'ont que les données qui leurs sont spécifiques, c'est-à-dire des informations qui ne décrivent pas une personne. La table ***info-crédit*** est une extension du ***client***, elle permet de garder toutes les informations sur les cartes de crédits liées aux clients. La table ***films*** possède toutes les données spécifiques aux films tout en faisant référence aux tables ***pays***, ***annonce***, ***genre*** et ***scénariste*** pour les informations externes aux films. Les tables ***loué***, ***tourné*** et ***catégorie*** sont créés afin d'éliminer les relations de plusieurs à plusieurs.

4. IMPLÉMENTATION et ALGORITHMES

Dans le but d'appliquer les bonnes pratiques de la programmation, une attention particulière a été apportée sur les processus de récolte et de validation des données avant d'entamer une connexion à la base de données. Dans le même ordre d'idées, du temps a été consacré pour bien segmenter les classes Java afin de supporter le développement ou la conception durables de l'application tout au long de la session.

Structure des packages java et pré-validation

Dans un premier temps, l'application se voit être composée de quatre ''packages''. Celle qui porte le nom de ''application'' contient les classes les plus vitales au bon fonctionnement du système. En effet, le démarrage de l'application, la lecture des fichiers xml, la validation des données, la connexion à la base de données, la lecture du script SQL et l'insertion des données dans la base de données sont regroupés dans le même ''package''. Ensuite, les ''packages'' nommés ''model'', ''gestion'' et ''database''servent principalement à supporter les classes vitales de l'application. Ces classes s'occupent principalement de garder en mémoire les données, d'afficher l'information dans la console, de gérer les caractères qui pourraient causer des conflits avec l'insertion des données dans la base de données.

La première partie concrète de l'implémentation démarre avec la classe ''LectureXML'' fournie par l'enseignant qui permet de récupérer toutes les valeurs et attributs des fichiers ''films.xml'', ''personnes.xml'' et ''clients.xml''. Pour chacun de ces fichiers, une méthode lui est attitrée. Pour chaque élément traité comme une personne, un film ou un client, les données sont envoyées vers la classe ''ValidationData'' afin de procéder à des étapes de vérifications qui consistent à repérer les anomalies et assurer la justesse des quantités. Pour arriver à supporter la récolte et sauvegarde des données, il a été nécessaire de créer des classes modèles que l'on retrouve dans le ''package'' ''model''. Éventuellement, ce processus de récolte et de pré-vérification qui se doit d'être supporté par des classes modèles ne sera plus nécessaire. Cette prise de décision avait comme objectif d'assurer à tout prix la justesse des données afin d'éviter une mauvaise orientation de conception.

Exécution script SQL

Ensuite, lorsqu'arrive le temps d'insérer les données dans la base de données, une connexion est initialisée avec l'information d'authentification fournie au début du projet. Il faut souligner qu'avant de procéder à l'insertion, on appelle une méthode qui pointe et démarre un script SQL qui s'occupe de retirer les tables (DROP TABLE) de la base de données créée antérieurement à partir d'un autre script SQL et de les remettre en place (CREATE TABLE). Cette méthode qui porte le nom ''executeScriptSQL'' est supportée par la classe ''ScriptRunner'' qui a été créée pour le projet ''Apache iBATIS''. L'appel du script avant l'insertion de données évite l'apparition de messages d'erreurs et assure un démarrage de l'application sur une base stable.

Syntaxe SQL

Pour insérer des données dans la base de données à partir de l'application java développée, il faut utiliser une syntaxe spécifique. En effet, il faut d'abord spécifier la table à traiter et pointer les colonnes qui recevront les données. Ci-dessous, vous retrouverez un simple exemple de la syntaxe à utiliser pour insérer des données dans la table INFOCREDIT.

```
String requeteCredit =
        "INSERT INTO INFOCREDIT (CARTE,NOCARTE,EXPMOIS,EXPANNEE,IDCLIENT)" +
        " VALUES (' '" + carte + " ',' " + no + " ',' " + expMois + " ',' " + expAnnee + " ' )";
```

Ci-dessous, vous retrouverez un simple exemple de la syntaxe à utiliser pour convertir un ''String'' (anniversaire) en type de donnée DATE.

```
String anniversaire = "TO_DATE ('" + anniv + "','YYYY-MM-DD')";
```

Ci-dessous, vous retrouverez un simple exemple de la syntaxe à utiliser pour récupérer une valeur d'une table à partir d'une valeur d'une autre table.

```
"INSERT INTO SCENARISTES (IDPERSONNE,IDFILM) SELECT IDPERSONNE," +
      idFilm + " FROM PERSONNES WHERE NOM = ' " + getScenaristes().get(i) + " ' ";
```

Type de donnée CLOB

Parmi les insertions effectuées dans la base de données, l'une d'entre elles requiert l'utilisation d'un type de donnée spécifique pour supporter la mise en mémoire d'un fort volume de caractères. La table PERSONNE possède une colonne BIO qui requiert justement ce type de donnée que l'on nomme CLOB. Pour arriver à transférer la totalité du texte vers la base de données, il faut le séparer en plusieurs blocs de ''String'' de moins de 3000 caractères et utiliser la méthode d'Oracle ''TO_CLOB()''. Les blocs inutilisés seront invisibles dans la base de données. Vous trouverez ci-dessous une version simplifiée du code implémenté pour supporter l'insertion du texte dans la colonne BIO de la table PERSONNE.

```
String bloc1 = "", bloc2 = "", bloc3 = "", bloc4 = "", bloc5 = "";

    for(int j = 0; j < personnes.get(i).getBio().length(); j++){
        if(j <= 3000)                    bloc1 += Character.toString(personnes.get(i).getBio().charAt(j));
        if(j > 3000 && j <= 6000)        bloc2 += Character.toString(personnes.get(i).getBio().charAt(j));
        if(j > 6000 && j <= 9000)        bloc3 += Character.toString(personnes.get(i).getBio().charAt(j));
        if(j > 9000 && j <= 12000)       bloc4 += Character.toString(personnes.get(i).getBio().charAt(j));
        if(j > 12000 && j <= 15000)      bloc5 += Character.toString(personnes.get(i).getBio().charAt(j));
     }

return ("TO_CLOB('" + bloc1 + "') || TO_CLOB('" + bloc2 + "') ||
          TO_CLOB('" + bloc3 + "') || TO_CLOB('" + bloc4 + "') || TO_CLOB('" + bloc5 + "')");
```

5. DISCUSSION

Points forts et les points faibles

Points faibles

1. Plusieurs membres de l'équipe n'ont pas une connaissance avancée dans les BD relationnels. Ce point connait une amélioration au fur et à mesure que nous approfondissons les théories dans le cours.

2. Pour s'assurer de la bonne orientation du projet, des classes modèles et des listes dynamiques sont utilisées pour effectuer une pré-validation (quantité & anomalies). Cela réduit la performance de l'app (surtout s'il faut imprimer les résultats dans la console pour la période de test)

Points Forts

1. L'application possède un menu utilisateur qui permet soit de démarrer une lecture locale des fichiers XML et visualiser les données dans la console, soit de créer une connexion à la BD pour insérer des requêtes manuelles, soit démarrer l'insertion de données automatique (selon le choix du fichier à traiter). Cela permet de tester de plusieurs façon l'app et supporter tous les membres dans leur apprentissage.

2. Pour chaque objet traité (film, personne, client), si un champ est vide, on s'assure d'insérer par exemple un String ''n/a'' pour éviter de se poser la question si le champ a bien été traité par l'app java. Si dans la DB on retrouve un champ vide, c'est qu'il y a un problème.

Motivation

Les premiers travaux de modélisation étant abstraits, ils apparaissent à première vue un peu difficile, car on devait construire un modèle conceptuel qui s'adapte à la description mentionnée dans les fichiers xml. Pour compléter ce travail, nous avons organisé un minimum de deux (2) réunions d'équipe et on travaillait séparément sur la compréhension du modèle.

Les autres difficultés rencontrées dans ce Labo, comme certains membres de l'équipe qui ne maîtrisent pas certains types de langages et/ou plateformes utiles pour la réalisation du labo. On a organisé les tâches, puis on s'entraide via des outils de travail collaboratif, et puis on avance en attribuant les tâches exactes aux coéquipiers qui ont un minimum de compétence dans son exécution. Ce faisant, nous trouvons que toute l'équipe est bien motivée afin que nous puissions développer un produit qui correspondra aux exigences telles que définies dans le cahier de charges.

Justification des choix de conception et d'implémentation

Nous avons choisi de développer le produit dans l'environnement Java, car à notre avis, ce dernier étant un produit de l'entreprise d'Oracle, l'intégration de la solution ne va pas être trop problématique. Qui plus est, beaucoup d'Environnements de Développement Intégrés de java, comme la plateforme eclipse, sont des logiciels libres. Les solutions libres offrent de nombreux avantages, comme le fait de pouvoir contrôler ce qu'on fait sans trop dépendre de l'éditeur du produit. Il s'ensuit aussi que nous utilisons les applications utilisées dans les labo pour la création de la BD, la définition et la manipulation des données. Pour la conception des modèles. Enfin, nous avons utilisé draw.io qui nous permet de collaborer à distance via un dossier google drive partagé que nous avons créé pour l'équipe.

Choix alternatifs

On pourrait envisager le développement dans d'autres environnements propriétaires comme celui de microsoft et d'autres environnement libres python par exemple. Cela risque de compliquer le travail, car il nous faudra plus de temps pour nous adapter à ces choix alternatifs. On développe en utilisant les méthodes les plus simples, mais nous nous assurons de fournir le travail tel que demandé.

Les améliorations possibles

Tout en soumettant ce premier TP, nous espérons recevoir les feedbacks nécessaires afin de pouvoir améliorer les modèles, au besoin. Les premier résultats de l'exécution du programme viennent avec une interface en mode console, ce qui fera l'objet d'autres modifications pour avoir le mode graphique dans l'application dans les prochains labos.

6. CONCLUSION

Durant ce laboratoire nous avons appris à mettre en place un système de base de données multimédia; pour cela, nous avons commencé par l'analyse des données contenues dans les fichiers xml, ensuite élaborer le modèle conceptuel qui a permis de définir les classes, le type, la multiplicité des attributs et les relations; et élaborer le modèle relationnel qui met en évidence les différentes tables, les clés primaires et étrangères ainsi que la relation entre elles puis leurs multiplicités.

L'implémentation nous a permis de nous familiariser avec les scripts SQL entre autres la création, la suppression, la sélection des données en résumé la manipulation des données de la base des données. L'objectif n'est pas complètement atteint, car nous avons rencontré plusieurs bugs que nous n'avons pas pu tout régler. Tout de même, nous avons réussi à développer le petit programme avec un menu pour pouvoir gérer la base des données.

7. RÉFÉRENCE

1. Conception des bases de données : Developpez.com (2019),
   Base de données et langage SQL,
   https://laurent-audibert.developpez.com/Cours-BD/?page=conception-des-bases-de-donnees-modele-a

2. Bases de données : Yousra Lembachar (2019),
   Le modèle relationnel,
   http://alumni.cs.ucr.edu/~ylemb001/chap2.pdf

3. w3schools.com (2019)
   SQL Tutoriel,
   https://www.w3schools.com/sql/