Mini-projet de Bases de Données

Kévin Vythelingum
Mohammed Amine Daouma

12 mai 2013

Table des matières

Introduction

L'objectif de ce projet est de réaliser un système d'information informatisé pour une bibliothèque municipale. Ainsi, la plupart des tâches de gestion de cette bibliothèque pourra être automatisé. Tout d'abord, la mise en place du système d'information demande une étape de modélisation afin d'identifier les contraintes et les données à représenter. Ensuite, à partir de ce modèle, nous réaliserons un schéma de base de données qui sera implémenté sous MySQL. De plus, nous entrerons dans cette base quelques données qui permettrons de simuler le fonctionnement de la bibliothèque. Pour finir, même si cette base est accessible depuis le client MySQL par défaut, nous réaliserons différents programmes en langage C qui serviront d'interface avec la base, d'une part pour les utilisateurs de la bibliothèque avec un outil de recherche, d'autre part pour les administrateurs avec un outil de gestion. Différents tests, qui seront expliqués dans ce compte-rendu, serviront à valider les fonctionnalités du projet à chaque étape de la réalisation.

Le projet est organisé de la façon suivante :

- Le dossier *Donnees* qui contient les fichiers de données textes qui peuplent la base initialement.
- Le dossier *Programmes* qui contient le code source des applications en langage C.
- Le dossier *SQL* qui contient le code des différentes composantes de la base. Le fichier init.sql permet de charger l'ensemble des autres fichiers dans la base.
- Le dossier *WinDesign* qui comporte le MCD et le MLR générés par WinDesign.

Chapitre 1

Modélisation

1.1 Analyse du problème

Telle que nous est présentée la bibliothèque, nous réaliserons d'abord un schéma entité-association que nous traduirons en modèle relationnel, modèle qui permettra l'implémentation sous MySQL. À la lecture des spécifications, deux entités paraissent naturelles et inévitables :

- les livres
- les abonnés

En effet, la première nécessité de la bibliothèque est d'avoir un catalogue de livres et une liste d'abonnés susceptibles d'emprunter ces livres. Or, ces livres peuvent être disponibles en plusieurs exemplaires différents qui ont le même titre et le même contenu mais qui peuvent provenir d'éditeurs différents par exemple. Nous différencierons donc *livre* et *exemplaire*.

De même, chaque livre peut être écrit par un ou plusieurs auteurs différents et sont d'un ou plusieurs genres différents (Roman, Bande dessinée, Policier, Romantique, Revue, etc.). Cela nous amène à ajouter des entités à celles déjà identifiées :

- les exemplaires de livre
- les auteurs
- les genres

De plus, la bibliothèque souhaite conserver un historique des emprunts effectués. Il ne faut donc pas supprimer le n-uplet correspondant lors du retour d'un livre par un abonné. Or, un abonné ne peut emprunter plus de 5 livres à la fois. Nous devons donc ajouter une entité *date* qui permettra la distinction entre les emprunts en cours et les emprunts passés d'un abonné. Cependant, cette entité ne correspondra pas à une table à part entière dans la base puisqu'elle n'a aucune réalité. Ce sera simplement un attribut ajouté à la table des emprunts.

En somme, notre modèle conceptuel comporte les entités suivantes :

- livre
- exemplaire
- abonné
- auteur
- genre
- date

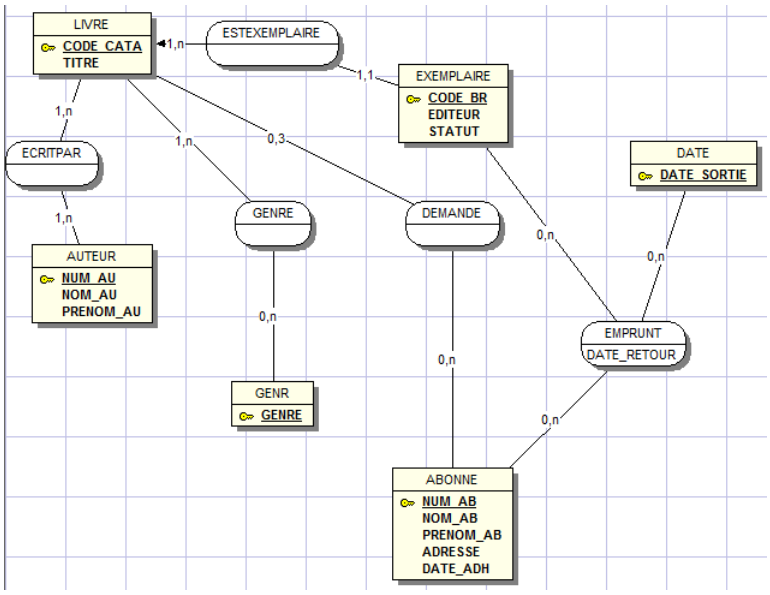
Il faut maintenant établir les relations entre ces différentes entités.
Tout d'abord, nous relions l'entité livre à :
– l'entité *exemplaire* par la relation *est_ exemplaire*
– l'entité *auteur* par la relation *écrit_ par*
– l'entité *genre* par la relation *de_ genre*
– l'entité *abonne* par la relation *demande* qui correspond aux demandes de réservation

Enfin, la relation *emprunt* relie les entités *exemplaire*, *abonné* et *date* et permet de gérer les emprunts comme son nom l'indique. Elle possède un attribut, *date-retour*, qui sera indéfini la plupart du temps et prendra la valeur de la date de retour du livre par un abonné lorsqu'il y a lieu.

Les cardinalités liées à ces associations ainsi que le détail des attributs des entités figurera sur le schéma entité-association.

1.2 Schéma Entité-Association

Le schéma entité-association est le résultat de l'analyse précédente. Nous avons cependant renommé l'association *de_ genre* en `GENRE`, et l'entité *genre* en `GENR` puisque cette dernière ne sera pas générée pour les mêmes raisons que `DATE`. Les cardinalités sont pour la plupart imposées par le cahier des charges. On remarque que l'on peut représenter sous forme de cardinalité le fait qu'un abonné ne peut formuler plus de trois demandes. Cependant, on ne peut pas faire de même pour la contrainte selon laquelle un abonné ne peut pas emprunter plus de cinq livres car on garde un historique des emprunts. Dans le modèle physique, ces deux contraintes seront satisfaites au moyen de triggers.



Remarque. L'attribut *statut* de l'entité *exemplaire* peut sembler superflu dans la mesure où il contient l'information *disponible*, *sorti*, *de côté*, ... qui peut être calculée. Néanmoins, bien que ce calcul soit compensé par un calcul de vérification d'intégrité lors d'une modification de la table *exemplaire*, le fait

de laisser cet attribut permet de stocker des informations sur l'état du livre s'il ne peut pas être emprunté (perdu, abîmé, etc.) De plus, l'existence de cet attribut *statut* offre une certaine facilité d'affichage du statut du livre lors d'une recherche utilisateur.

1.3 Schéma Relationnel

À partir du schéma entité association, on peut obtenir le schéma relationnel par simple traduction. Le schéma est donc le suivant :

- livre (code-catalogue, titre)
- exemplaire (code-barre, code-catalogue, éditeur, statut)
- abonné (n-abonné, nom, prenom, adresse, date-adhésion)
- auteur (n-auteur, nom, prenom)
- genre (genre, code-catalogue)
- date (date-sortie) *Cette table n'est pas représentée dans le modèle physique*
- écrit_par (n-auteur, code-catalogue)
- demande (n-abonné, code-catalogue)
- emprunt (n-abonné, code-barre, date-sortie, date-retour)

Certaines contraintes ne sont pas captées par le schéma. Il faudra donc les vérifier à chaque insertion de nouvelles valeurs au moyen de triggers ou de vues spécifiques. Ainsi, il faudra assurer que :

- un abonné ne peut pas formuler plus de trois demandes
- un abonné ne peut pas emprunter plus de cinq livres à un instant donné
- un même livre ne peut pas être emprunté en même temps par plusieurs abonnés
- un livre *de côté* ne peut être emprunté que par des abonnés ayant effectué une demande sur ce livre
- un renouvellement ne peut être effectué que s'il n'y a pas de demande sur le livre concerné

Remarque. «On ne gère pas les priorités sur les demandes» Cette hypothèse simplificatrice pose un problème au niveau des emprunts. En effet, si un abonné qui n'a pas effectué de demande préalable souhaite emprunter un livre dont le statut est *de côté* car il a été réservé par d'autres abonnés, il a juste à effectuer une demande de réservation sur ce livre afin d'être ajouté à la liste des demandeurs. Par cette opération, il peut contourner le système des réservations et emprunter le livre désiré immédiatement. Ceci pose un problème dans l'utilité même des réservations. Une solution à ce problème serait d'interdire la formulation de nouvelles demandes pour un livre dont le statut est *de côté*. Cependant, dans le cadre de l'exercice, il a été convenu de ne pas prendre en compte ce problème.

1.4 Formulation en SQL

Le code SQL de base est généré automatiquement par le logiciel WinDesign à partir du schéma entité-association. À ce code, il faut ajouter après chaque définition de table `ENGINE=InnoDB` afin de prendre en compte les clefs étrangères

et le système transactionnel. Il faut demander à ne pas générer la table *date* et la table *genr* qui n'ont pas de réalité physique.

Ensuite, il faut écrire les triggers, les procédures stockées et les vues.

1.4.1 Triggers

Les triggers, déclenchés lors de certaines requêtes sur la base de données, veillent à ce que celle-ci reste dans un état cohérent.

Lors d'une nouvelle demande de réservation

À chaque nouvelle demande, c'est-à-dire une insertion dans la table `DEMANDE`, on vérifie que l'abonné en question n'a pas déjà effectué au moins trois demandes. S'il a déjà atteint son quota de trois demandes, on génère une erreur.

Lors d'un nouveau prêt

Lors d'un nouveau prêt, c'est-à-dire une insertion dans la table `EMPRUNT` :
- que l'abonné n'a pas déjà emprunté 5 livres
- que le livre désiré n'est pas déjà l'objet d'un emprunt par un autre abonné
- que le livre n'est pas mis de côté alors que l'abonné qui veut l'emprunter n'est pas sur la liste d'attente

Si l'un des trois cas se présente, on génère une erreur. Sinon, on met à jour le statut du livre à *sorti*. Dans le cas d'un emprunt de livre réservé, on supprime la demande de l'abonné de la liste d'attente des réservations.

Lors d'un renouvellement

Lors d'un renouvellement, c'est-à-dire une mise à jour de la table `EMPRUNT` pour un n-uplet dont la date de retour est indéfinie, on vérifie que le livre à renouveler n'est pas réservé. Si c'est le cas, on génère une erreur.

Lors d'un retour

Lors d'un retour, c'est-à-dire une mise à jour de la table `EMPRUNT` avec une date de retour non nulle, on vérifie s'il existe des demandes ou non sur le livre rendu afin de mettre son statut à *de côté* ou *disponible*.

Remarque. Pour générer une erreur, on réalise une action illicite pour la base de donnée, comme par exemple donner la valeur `NULL` à un attribut qui ne peut pas l'être. Ainsi, une erreur est remontée par le système et donc l'utilisateur est prévenu que sa requête est impossible.

1.4.2 Procédures stockées

Les procédures stockées vont permettre de faciliter certaines actions récurrentes afin de ne pas les détailler à chaque appel. En outre, cela permet de cacher le détail de certaines fonctions qui seront utilisées dans les programmes en C.

Retrait d'un abonné

Lorsque l'adhésion d'un abonné expire, on le supprime de la base de données en effectuant séquentiellement :

- le retrait de ses demandes dans la table `DEMANDE`
- le retrait de ses emprunts dans la table `EMPRUNT`
- le retrait de l'abonné lui-même dans la table `ABONNE`

Nouvelle inscription

Lors d'une nouvelle inscription, on effectue une insertion dans la table `ABONNE` en précisant numéro, nom, prénom et adresse de l'abonné en paramètre. La date d'adhésion est automatiquement prise en compte grâce à la fonction `CURDATE()` qui renvoie la date courante.

Gestion des livres en attente

Cette procédure intitulée `supprimerLivreAttente()` permet de supprimer les demandes en attente sur un livre lorsque cela fait plus d'une semaine qu'un exemplaire de ce livre est de côté. De plus, cet exemplaire est rendu disponible.

Pour cela, on supprime d'abord toutes les demandes concernant des livres dont un exemplaire est de côté depuis au moins une semaine, puis on passe à *disponible* les exemplaires de côté qui ne font plus l'objet d'une demande.

1.4.3 Vues

Pour faciliter l'édition de certaines listes, différentes vues seront réalisées.

Vue catalogue

La vue catalogue représente l'ensemble des exemplaires de livres détenus par la bibliothèque, chacun accompagné de détails informatifs. Cette vue sera celle utilisée pour effectuer des recherches au niveau utilisateur. Elle consiste en une jointure des différentes tables concernant le catalogue de la bibliothèque. L'énoncé est le suivant :

```
select CODE_BR,TITRE,PRENOM_AU,NOM_AU,EDITEUR,GENRE,STATUT
from EXEMPLAIRE,LIVRE,AUTEUR,GENRE,ECRITPAR
where EXEMPLAIRE.CODE_CATA=LIVRE.CODE_CATA
      and LIVRE.CODE_CATA=ECRITPAR.CODE_CATA
      and ECRITPAR.NUM_AU=AUTEUR.NUM_AU
      and LIVRE.CODE_CATA=GENRE.CODE_CATA
```

Il suffira de préciser `where TITRE="<titre>"`, etc. pour affiner la recherche.

Vue retardataires

La vue retard permet d'éditer la liste des numéros et noms des retardataires, qui devront payer une amende de un euro lors du retour du livre détenu. Les retardataires ainsi listés sont accompagnés du code barre et du titre des livres qu'ils auraient dus rendre. L'énoncé est le suivant :

```
select EMPRUNT.NUM_AB,PRENOM_AB,NOM_AB ,EMPRUNT.CODE_BR,TITRE
from EMPRUNT,LIVRE,ABONNE,EXEMPLAIRE
where DATE_RETOUR=0000-00-00
      and DATEDIFF(CURDATE(),DATE_SORTIE)>15
      and EMPRUNT.NUM_AB = ABONNE.NUM_AB
      and EMPRUNT.CODE_BR = EXEMPLAIRE.CODE_BR
      and EXEMPLAIRE.CODE_CATA = LIVRE.CODE_CATA;
```

1.5 Données et tests

Pour simuler le fonctionnement de la bibliothèque, nous avons inséré des données dans chaque table en veillant à avoir le maximum de cas possibles qui peuvent arriver au cours de la vie du système d'information. Afin de réaliser ceci, nous avons entré les données de chaque table dans une feuille de calcul d'un tableur que nous avons exporté au format texte/csv. Ces fichiers texte ont ensuite été insérés dans la base de donnée par la commande SQL `load`. Afin de faciliter le chargement des données, et en prévision de modification des données à charger, nous avons regroupé les commandes de chargement dans un fichier *loaddata.sql*. Ce fichier comporte deux parties :

- Une première partie de suppression des données de la base basée sur des commandes `delete`
- Une deuxième partie de chargement à proprement parlé des données basée sur des commandes `load`

Les tests de cette partie ont tout d'abord consisté en la vérification que toutes les données entrées aient bien été prises en compte. Cela s'est fait par une suite de commande `select * from ...` sur chaque table. Ensuite, nous avons effectuées diverses insertions et suppressions afin de vérifier que nos contraites d'intégrités étaient toujours vérifiées. Par exemple, nous avons tenté de faire emprunter plus de 5 livres à un abonné afin de voir si une erreur était bien déclenchée, etc. Aussi, nous avons vérifié que nos procédures stockées effectuaient bien les actions attendues et que nos vues éditaient bien les listes demandées.

Chapitre 2

Menu utilisateur

2.1 Objectif

Le but du menu utilisateur est de proposer à tous un outil de recherche dans le catalogue de la bibliothèque. Grâce à ce programme, abonnés ou non pourront effectuer des recherches aussi bien par nom d'ouvrage, par nom d'auteur, par nom d'éditeur ou par genre. Dans tous les cas, les informations délivrées seront sous la forme d'une liste comportant code barre, titre d'ouvrage, nom d'auteur, éditeur et statut.

2.2 Fonctionnement

Le programme fonctionne de manière à guider l'utilisateur au maximum en lui masquant les détails d'implémentation de la base de donnée. Pour cela, il lui est demandé de choisir son type de recherche. Selon son choix, la procédure `recherche` sera exécutée de manière différente. Celle-ci prépare la requête dans un premier temps, puis l'exécute avant de lancer la procédure `afficherResultat` qui met en forme le retour du serveur MySQL.

2.3 Compilation

Le menu utilisateur se compile à l'aide de la commande suivante :
`gcc -o menu-utilisateur utilisateur.c -lmysqlclient`

2.4 Tests

Nous avons testé tous les types de recherche en utilisant en particulier des casses différentes et en recherchant des auteurs homonymes, ainsi que des livres à exemplaires, auteurs ou genres multiples. Il est également possible de rechercher des noms d'ouvrages comportant des espaces. Il est important de noter que plusieurs lignes concernant le même exemplaire peuvent être affichées si l'exemplaire en question possède plusieurs genres et/ou auteurs. Nous avons considéré que ce problème n'était pas grave dans la mesure où l'utilisateur était à même de différencier les exemplaires par leur code barre qui est affiché. De

plus, un `select distinct ...` aurait induit une perte d'information. Ces différents tests ont permis de valider notre programme.

Chapitre 3

Menu administrateur

3.1 Objectif

Le but du menu administrateur est de proposer quatre procédures principales que sont la Gestion des prêts et des demandes, la Mise à jour du fichier des abonnés, la Gestion des livres en attente et l'Echéancier. Le programme est globalement très similaire au menu utilisateur dans son fonctionnement, la différence consistant simplement en des fonctionnalités plus avancées.

3.2 Gestion des prêts et des demandes

La procédure de gestion des prêts et des demandes, traitée par la procédure `prets_et_demandes`, permet de réaliser quatre actions distinctes :

- Enregistrer un nouveau prêt
- Enregistrer une nouvelle demande de réservation
- Enregistrer un retour
- Enregistrer un renouvellement

Le choix effectué est passé en argument de la procédure `prets_et_demandes` qui va effectuer la requête demandée et afficher s'il y a succès ou échec de celle-ci. Lors de l'enregistrement d'un retour, un message s'affiche en cas de retour tardif indiquant à l'abonné de payer une amende de un euro.

3.3 Mise à jour du fichier des abonnés

La procédure de mise à jour du fichier des abonnés, traitée par la procédure `maj_fichier_abonne`, permet de réaliser trois actions distinctes :

- Supprimer les abonnés de la base lorsque leur adhésion a expiré
- Enregistrer une nouvelle inscription (il est possible d'insérer des espaces pour le nom, le prénom et l'adresse du nouvel abonné)
- Renouveler une inscription

Le choix effectué est passé en argument de la procédure `maj_fichier_abonne` qui va effectuer la requête demandée et afficher s'il y a succès ou échec de celle-ci.

3.4 Gestion des livres en attente

La procédure de gestion des livres en attente, traitée par la procédure `gestion_livre_attente`, va demander si oui ou non on souhaite exécuter la procédure stockée `supprimerLivreAttente()`. Dans le cas d'une réponse positive, on affichera si le traitement de la requête est un succès ou un échec.

3.5 Echéancier

La procédure `echeancier` édite simplement la liste des retardataires avec, pour chacun, la liste des livre qu'il aurait du rendre. Cela consiste à mettre en forme le résultat d'un `select * from RETARD`.

3.6 Compilation

Le menu administrateur se compile à l'aide de la commande suivante :
`gcc -o menu-administrateur administrateur.c -lmysqlclient`

3.7 Tests

Nous avons testé toutes les fonctionnalités du menu administrateur en vérifiant que les contraintes d'intégrités étaient toujours respectées et que les demandes illicites étaient bien refusées. Nous avons notamment effectué des retours normaux (cb004) et tardifs (cb012) pour vérifier qu'un message s'affiche bien au cas où l'abonné doit payer une amende.

Conclusion

En somme, nous sommes parvenus à répondre au cahier des charges en fournissant une base de donnée complète et deux clients écrits en langage C qui permettent d'interagir avec la base. Ce projet nous a appris à un logiciel de modélisation, WinDesign, dont le fonctionnement était proche de la méthode apprise en cours pour réaliser "à la main" un schéma SQL à partir d'un schéma entité-association. De plus, nous avons pu mettre en oeuvre des fonctionnalités avancées de MySQL comme les vues, les triggers et les procédures stockées.

Dans l'ensemble du projet, nous nous sommes efforcés de mettre en place au maximum les vérifications des contraintes d'intégrité et les procédures au niveau de la base MySQL. Cela permet d'une part de faciliter la réalisation des différents programmes en langage C et d'autre part d'offrir une meilleure cohérence avec de possibles clients web ou mobile. En effet, si les contraintes étaient vérifiées au niveau du programme client, il faudrait s'assurer que tous les programmes vérifient les mêmes contraintes.

Pour aller plus loin, il aurait été possible de réaliser des outils de gestion des livres permettant d'ajouter de nouveaux ouvrages à la bibliothèque, de spécifier de nouveaux genres, de nouveaux auteurs, etc. Il serait également utile d'exploiter davantage l'attribut *statut* de la table *exemplaire* en proposant des règles supplémentaires liées aux statuts *abîmé*, *perdu* ou encore *en consultation uniquement* qui pourrait être appliqué aux dictionnaires et encyclopédies par exemple.