Laboratoire d'Analyse et gestion de données

Application fichiers en langage C

« QuizUp_Like »

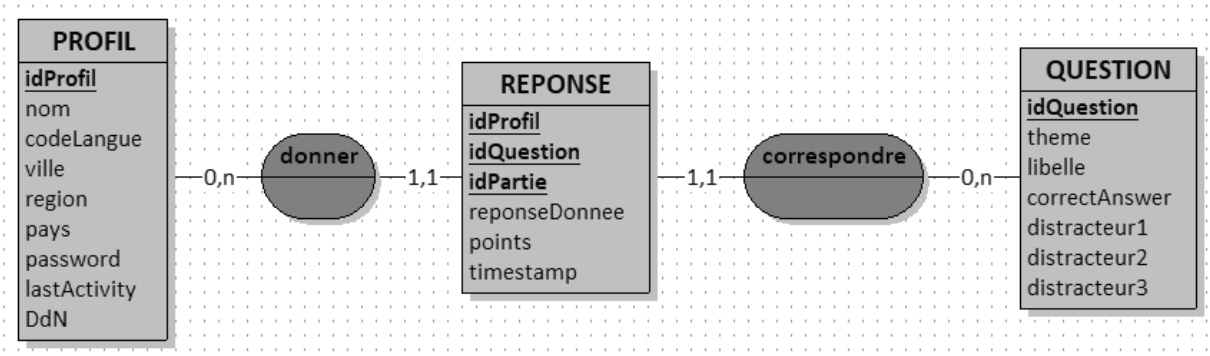
Guide pour la réalisation du projet

Situation au 14/02/2019

Table des matières

But :

Nous devons concevoir une application pour réaliser un gestionnaire de base de données dans le but de trouver une solution au MRD suivant :



Vue d'ensemble de l'objectif :

Pour réaliser notre application, nous aurons une organisation physique des fichiers de 3 manières différentes :

- Organisation séquentielle physique pour les profils (fichier : `profils.dat`)
- Organisation séquentielle logique pour les réponses (fichier : `reponses.dat`)
- Organisation indexée directe pour les questions (fichier : `questions.dat`)

Pour se faire nous aurons un ensemble de contrainte d'intégrité référentielle entre profils, réponses et questions ainsi que des contraintes à l'encodage et à l'affichage.

Réalisation du projet :

Vous trouverez ci-dessous une proposition de ligne de conduite pour réaliser le projet. Ceci consiste juste en des suggestions pour vous guider dans le cheminement de la conception du programme, chaque personne est libre de le faire suivant sa façon de penser.

Partie 1 : les contraintes.

Pour l'encodage et l'affichage, nous avons une série de contrainte à respecter. Nous pouvons réaliser un ensemble de fonction qui permet de gérer celle-ci de manière à pouvoir les appeler dès que le besoin s'en fera ressentir. Il n'est parfois pas inutile de regarder si des fonctions ne sont déjà pas réalisées dans des librairies de C (exemple : « ctype.h »).

Voici un exemple de site qui vous donnera des références pour une série de librairies de C disponible : https://www.tutorialspoint.com/c_standard_library/index.htm

Liste des fonctions proposées pour l'application des contraintes :

- existeFichier(char *Nomfichier) : permet de savoir si un fichier existe, permettra de contrôler si nos fichiers profils.dat, reponses.dat et questions.dat existe avant de les créer.
- retireEnter(char *Chaine) : permet de retirer un caractère de retour à la ligne (ENTER) en fin de chaine. Attention, une chaine doit toujours avoir un marquage de fin (char 0).

- valideLettre(char *Chaine) : permet de vérifier qu'une chaine ne contient que des caractères de type a,z et A,Z.
- valideCaractere(char *Chaine) : permet de contrôler les caractères de type . , '-_ et espace.
- valideImprimable(char *Chaine) : permet de vérifier qu'une chaine ne contient que des caractères imprimables.
- valideChiffre(char *Chaine) : permet de vérifier qu'une chaine ne contient que des chiffres de 0 à 9.
- agePersonne(int *Age, date *datenaissance) : permet de savoir l'age suivant une date de naissance.
- Initiale(char *Chaine) : permet de mettre la majuscule à l'initiale des une chaine de caractère (la première lettre en majuscule et les lettres suivants un espace en majuscule)
- Majuscule(char *Chaine) : permet de mettre une chaine de caractère en majuscule.
- Minuscule(char *Chaine) : permet de mettre une chaine de caractère en minuscule.
- Duree(int *seconde, time_t *tdebut, time_t *tfin) : permet de calculer le nombre de seconde entre tdebut et tfin.

Dans certaines contraintes, ils nous suffiront de composer avec plusieurs de nos fonctions. Exemple pour le contrôle de chaines de caractères qui ne peuvent avoir que des chiffres et des caractères.

Partie 2 : les structures.

Nous avons dans notre application les structures suivantes qui ne peuvent être changées :

- Profil
- Reponse
- Question
- Date

Pour faciliter la réalisation du programme, il serait judicieux de faire des fonctions pour manipuler ces structures. Pour la gestion du contrôle de l'intégrité référentielle lors de l'encodage de certaine structure, celle-ci ne pourront se faire seulement quand nous aurons géré l'encodage dans nos fichiers, nous ferons par conséquent des fonctions vides en attendant.

Voici la liste des fonctions qui vous est proposées de réaliser.

Pour la structure Date :

- encodeDate(struct date *madate) : permet l'encodage de la date.
- afficheDate(struct date *madate) : permet l'affiche de la date.
- …

Pour la structure Profil :

- encodeProfil(struct profil *leprofil) : permet l'encodage du profil.
- afficheProfil(struct profil *leprofil) : permet l'affiche du profil.
- …

Pour la structure Reponse :

- encodeReponse(struct reponse *mareponse) : permet l'encodage d'une réponse.
- afficheReponse(struct reponse *mareponse) : permet l'affiche d'une réponse.
- …

Pour la structure Question :

- encodeQuestion(struct question *maquestion) : permet l'encodage d'une question.
- afficheQuestion(struct question *maquestion) : permet l'affiche d'une question.
- …

Il est bien évident que cette liste de fonction peut être agrandie pour ajouter des éléments nécessaires suivant des contraintes spécifiques qui pourront prendre place dans une fonction pour en faciliter l'usage et la compréhension du code.

Partie 3 : les menus d'affichage

Nous allons réaliser les menus d'affichages pour la navigation des différentes parties de notre application. Il serait judicieux de faire une fonction pour chaque menu. Ce qui devrait nous conduire à avoir 4 fonctions pour les menus :

- menuGeneral()
- menuProfil()
- menuReponse()
- menuQuestion()

Rien ne nous empêche de réaliser des sous-menus pour certaines parties.

Partie 4 : gestion des fichiers

Pour cette partie, vous recevrez des librairies pour chaque type d'encodage des données qui vous permettront de réaliser la gestion des fichiers. Il vous faudra adapter certaine partie des librairies pour les rendre compatible avec nos structures de données.

Il est vivement conseillé de tester les libraires simplement (sans structure, juste avec un chiffre) avant de les adapter pour être sûr de bien en comprendre le fonctionnement. Regarder la taille de vos fichiers créés pour voir si elle est possible.

Librairie Organisation Physique

Pour cette librairie, il vous faudra adapter la structure « EnrSeqPhys » pour l'adapter aux bouteilles. La fonction « searchFileSeqPhys » devra être complétée pour répondre au besoin de votre structure. Nous vous avons conçu des librairies pour l'ajout, la modification, la création du fichier, la récupération d'un enregistrement et la recherche.

Dans cette librairie, il vous manque une fonction de suppression. Vous avez deux possibilités pour la réaliser. Vous pouvez faire une suppression logique ou physique. Bref, il vous faudra ajouter une fonction du style :

- deleteFileSeqPhys(char * monfichier, long position)

Librairie Organisation Logique

Pour cette libraire, il vous faudra adapter la structure « PayLoad » pour l'adapter aux fournisseurs. La fonction « comparePayLoad » devra être adaptée en fonction de votre structure et de votre critère de recherche. Nous vous avons conçu les librairies pour l'ajout, la suppression, la recherche et la création du fichiers.

Ces librairies sont volontairement incomplètes. Il vous manque une fonction pour la modification d'une structure du fichier et la fonction qui gère l'affichage. Il vous est demandé de les créer.

Il vous est conseillé de créer une fonction qui vous donnera les valeurs de PTO et de PTL de votre fichier fournisseur.

En résumé, nous vous proposons de créer les fonctions suivantes en plus de notre librairie.

- headerFileSeqLog(char * monfichier,Header * monheader)
- readFileSeqLog(char * monfichier,long position,PayLoad * record,long *suivant)
- updateFileSeqLog(char * monfichier, long position, PayLoad *record)

Astuce : pour la modification d'un enregistrement, si vous avez bien compris le principe du PTO et du PTL , il est très facile de faire une mise à jour, n'oubliez jamais que le PTL contient le dernier enregistrement supprimé et que par conséquent, le prochain ajout sera… 😉

Librairie Organisation Logique

Pour cette librairie, il vous faudra adapter les structures suivantes :

- EnrIdx
- IdxRecord

Ces librairies sont volontairement incomplètes. Vous avez les fonctions suivantes :

- int createFileIdx(char *, int);
- int allocateIdx(IdxRecord **, int);
- IdxRecord * buildIdx(char *);
- int insertIdx(IdxRecord *,IdxRecord *,int);
- int compareIdxEnreg(IdxRecord *,IdxRecord *);
- int writeFileIdx(char *,EnrIdx *);
- int readFileIdx(char *,EnrIdx *,long);
- int searchFileIdx(char *,char *,long,EnrIdx *);
- int updateFileIdx(char *,long, EnrIdx *);

Toutes ses fonctions ne sont pas complètes ou absentes. Il vous faudra les créer ou les adapter. Il est souhaitable de commencer par adapter la fonction de comparaison (compareIdxEnreg) en fonction de votre structure. Il vous faudra regarder l'énoncé pour savoir dans quel ordre effectuer les comparaisons pour respecter l'ordre dans votre index.

Lorsque le fichier questions.dat sera créé, comme vous avez un fichier avec un nombre fixe de case, celui-ci ne devra jamais changer. Sinon, il y a une erreur dans votre application. Pour savoir si votre taille est correcte, il suffit de diviser la taille du fichier par la taille d'un enregistrement et observer si le résultat correspond bien au nombre d'enregistrement demandé.

Remarque :

N'oubliez pas que vous avez un professeur qui est la lors des séances de laboratoire. Il est là pour répondre à vos questions et vous guidez lorsque vous vous sentez perdu dans la conception de votre application.