**Rapport**

**Projet de C : application de gestion d’une base de données**



*DELÉE Alexis*

*HOUNZA Brice*

*VECHOT Paul*

3ème année en Architecture des Logiciels, classe 2

M. Briatte

**Table des matières**

[I. Présentation du sujet 3](#_Toc500583699)

[II. Analyse de l’application 4](#_Toc500583700)

[a. Environnement 4](#_Toc500583701)

[b. Choix d’implémentation 4](#_Toc500583702)

[c. Structures de données 4](#_Toc500583703)

[d. Remarques 5](#_Toc500583704)

[III. Bilan 6](#_Toc500583705)

1. **Présentation du sujet**

A la fin de la semaine thématique de C, on nous a demandé de nous mettre en groupe de trois (Alexis Delée, Brice Hounza et Paul Vechot) afin de réfléchir et de développer en langage C une application de gestion d’une base de données relationnelle au format YAML.

Cette application devra inclure les fonctionnalités suivantes :

* Création / suppression d’une base de données
* Création / suppression d’une table sans limite
* Insérer / modifier / récupérer / supprimer des données
* Un parser SQL :
  + CREATE DATABASE / CREATE TABLE
  + DROP DATABASE / DROP TABLE
  + INSERT INTO / UPDATE / SELECT / DELETE FROM
  + HELP
  + EXIT
* Une interface d’administration graphique avec GTK

1. **Analyse de l’application**
   1. **Environnement**

Que ce soit sur Windows ou sur Linux, il est impératif d’avoir la variable d’environnement HOMEPATH ou ~ pointant vers votre dossier utilisateur.

Si vous êtes sur une version de Windows XP, vérifier bien que HOMEPATH valle bien \Documents and Settings\{username}

Le dossier racine des bases de données sera donc localisé dans $HOMEPATH/.yaml/

L’exécutable final se trouve dans bin/debug/yamlDB.exe, mais il est possible d’exécuter chaque couche indépendamment (avec ses dépendances) {couche}/manager/bin/debug/manager.exe

*L’exécutable peut être lancé de n’importe où tant que le fichier yaml.conf (ci-dessous) se trouve au même niveau.*

A la racine du projet se trouve aussi un fichier de configuration yaml.conf permettant de paramétrer certains comportements de l’application :

* allow-color (valeur par défaut : false – configurable dans Common/settings.c) : couleur syntaxique dans les shells supportant les séquences \033[XXm
* tree (valeur par défaut : false – configurable dans Common/settings.c) : affiche l’arbre à N branches généré lors du parsing SQL
* debug (valeur par défaut : false – configurable dans Common/settings.c) : active le mode debug, ce mode affiche plus d’informations en cas d’erreurs  
  Exception: this is an example of error  
   at \_\_FILE\_\_/function:\_\_LINE\_\_
  1. **Choix d’implémentation**

Nous avons décidé de partir sur une architecture de type « trois tiers ». Cette architecture nous semblait la plus adaptée étant donné qu’elle divise l’application en trois niveaux distincts de couches logicielles :

* La couche « Database » avec l’accès aux données persistantes (partie 1)
* La couche « Business » correspondant au traitement métier des données (parties 2 et 3)
* La couche « Services » pour l’affichage des données (partie 4)

Son gros avantage était dans la division des couches, elle nous permettait ainsi le développement de « modules » facile à intégrer et n’entrainant que très peu de conflits lors de merge sur git.

Un set de tests unitaires sur la partie Database a aussi été développé (Audit/Database.TU.c). Cette partie étant critique et soumise à des changements réguliers (améliorations, bugs…), il était vital de rendre la détection de problèmes le plus rapide possible.

* 1. **Structures de données**

Nous avons utilisé deux types de structures de données :

* Les « simples » pour des regroupements de données plus faciles à traiter.
* Les « classes ». Bien qu’il n’ait pas de notion de classe en C, nous avons simulé quelques propriétés de ce dernier, notamment le système de méthodes privées et publiques. Une méthode privée sera simplement une fonction sans son entête dans un .h (donc disponible uniquement dans le fichier .c dans lequel elle est créée), alors qu’une méthode publique sera un enchainement de pointeurs de fonctions dans une structure mère.

Les « classes » ont comme avantages de pouvoir regrouper plusieurs méthodes et sous-méthodes ainsi que d’aider à développer plus rapidement dans certains IDE grâce à une auto-complétions avancée (CodeBlocks, Clion…).

Les structures de données « simples » :

* Common/throw.c
  + struct Throw : gestion des exceptions levées
* Database/Yaml.c
  + struct Stack : « pile » avec les valeurs de retour du select
  + struct SelectStatement / struct whereStatement : regroupement des données de la recherche lors du select, update et delete
* Business/Parser.c
  + struct Node : branche de l’arbre

Les structures de données de type « classe » :

* Common/Interface.c
  + struct Interface : s’occupe de la gestion des interfaces de saisie dans le shell
* Database/Yaml.c
  + struct Yaml : gestion de la base de données
* Database/Entity.c
  + struct Entity : traitement sur les données brutes de la base de données
* Database/Engine.c
  + struct Engine : moteur de requêtage
* Business/Parser.c
  + struct Parser : analyse les requêtes SQL
* Business/Callback.c
  + struct Callback : gestion des fonctions appelées lorsqu’une requête est reconnue par le système
  1. **Remarques**

Nous avons ajouté une commande SQL spéciale pour récupérer des informations sur la table elle-même (TABLE\_NAME, COLUMN\_NAME et DATA\_TYPE) :

*select \* from INFORMATION\_SCHEMA.COLUMNS where TABLE\_NAME = database.table*

1. **Bilan**

Bien que Paul ne soit pas très à l’aise avec le langage C, Brice et moi avons pu bien avancer le projet en nous répartissant les tâches en fonction de nos capacités / connaissances.

Les points suivants n’ont pas pu être résolus :

* Select sur plusieurs tables en même temps
* Recherche plus complexe avec AND / OR

Pour finir à temps, il nous aurait fallu quelqu’un de supplémentaire.

Pour la très grande partie du projet, la norme C89 est respectée. C’est pourquoi on peut retrouver des notions de transtypages sur les structures à certains endroits du code. Toutefois cette notion est assez compliquée à comprendre mais Paul a fait de son mieux pour l’appréhender.

Distribution des tâches :

|  |  |  |  |
| --- | --- | --- | --- |
|  | Couche « Database » | Couche « Business » | Couche « Services » |
| Alexis Delée |  |  |  |
| Brice Hounza |  |  |  |
| Paul Vechot |  |  |  |

|  |  |
| --- | --- |
|  | Aucune participation |
|  | Participation faible |
|  | Participation importante |