1 Introduction

2 Stockage de la base de données

Afin de stocker les données de milliers d'étudiants, plusieurs choix se sont offerts à nous. Tout d'abord, nous avons utilisé std::map, comparable à un dictionnaire en python, dont les clés ont été l'identifiant de l'élève et les valeurs la struct student. Seulement, cela prenait beaucoup de place en mémoire et nous avons donc délaissé l'idée. Nous avons

3 Fin d'une query

Pour signaler la fin d'une query, nous avons implémenté une message queue grâce à une pipe. Lorsqu'on envoie une query à un processus, nous incrémentons une variable globale OperationInProgress qui indique le nombre d'opérations en cours. Lorsque qu'un processus termine sa query, il écrit dans une pipe. Le processus principal va lire cette pipe et va décrémenter la variable globale au fur et à mesure. Le pipe est non-bloquant, ce qui permet au processus principal de continuer même si il ne lit rien dans la pipe. Lors d'une transaction, on limite OperationInProgress à 1.

4 Classes

Nous avons mis la database_t et le query_result_t dans une classe, afin de rendre le code plus lisible. Par conséquent, nous avons implémenté un constructeur dans query_result_t, au lieu du query_result_init, pour que le code crée directement le query_result_t, sans devoir appeller l'initialisation à chaque création d'objet.

5 Améliorations possibles

5.1 std::map

Nous avions comme idée de stocker, dans un structure de données std::map (équivalent d'un dictionnaire en Python), les identifiants des étudiants en clés ainsi que leur indice de position dans la base de données en valeurs de ces clés.
Selon nous, cela aurait permis de faciliter et d'améliorer l'efficacité de la vérification de si un étudiant est déjà dans la base de données ou non.
En effet, lors de l'ajout d'un étudiant, nous aurions juste à vérifier si l'identifiant est présent dans le map pour éviter d'avoir des doublons dans la base de données.