Bases de Données Avancées : TP3

Au menu : le **BufferManager** (et ses tests !) :)

A. Information : gestion des buffers / de la RAM par le BufferManager

Comme pour le **DiskManager**, vous allez devoir coder l' « API » que le **BufferManager** offre aux couches plus hautes (c'est à dire, l'ensemble des méthodes que ces couches plus hautes vont appeler), mais aussi sa gestion interne.

Pour commencer, souvenez-vous (mais le cours est toujours disponible sur Moodle ! ;)) des éléments associés à chaque buffer :

- le PageId de la page qui s'y trouve chargée (si une page est chargée dans le buffer couramment)
- le pin_count
- le flag dirty

Les politiques de remplacement que vous allez devoir coder sont **LRU** et **MRU**. Suivant vos choix d'implémentation, ces politiques peuvent nécessiter le rajout d'autres informations associées aux buffers (par exemple, le temps).

Le choix initial d'une de ces politiques dépendra de ce que la **DBConfig** passée au constructeur spécifie (voir ci-dessous), toutefois un changement de politique « en cours de route » devra être géré proprement !

B. Code : rajout d'informations dans DBConfig

Rajoutez, dans la classe **DBConfig**, deux variables membres :

- *bm_buffercount*, correspondant au nombre de buffers gérés par le **BufferManager**
- *bm_policy*, correspondant à la politique de remplacement utilisée. Vous pouvez, en fonction de vos préférences et du langage utilisé, choisir pour représenter ce paramètre une variété de types (chaine de caractères, entiers, enums…).

Rajoutez également la prise en compte de ces nouveaux paramètres dans le constructeur ainsi que dans la méthode de chargement de la configuration depuis un fichier (voir TP1).
Modifiez et enrichissez vos tests de **DBConfig** pour prendre en compte ces rajouts.

C. Code : BufferManager

Créez une classe **BufferManager** comportant (au moins) les méthodes suivantes :

- Un constructeur qui prend en argument une instance de **DBConfig** et une instance de **DiskManager,** et qui stocke en variables membres une copie ou un pointeur / référence vers la config et *un pointeur / référence vers l'instance* de **DiskManager**. Attention à surtout ne pas copier / dupliquer l'instance de **DiskManager** !!!

- Une methode
  buff **GetPage** (*pageId*) avec *pageId* un **PageId** et *buff* un buffer.

Cette méthode doit répondre à une demande de page venant des couches plus hautes, et donc *retourner un des buffers gérés par le **BufferManager,*** rempli avec le contenu de la page désignée par l'argument *pageId*.

*Attention : ne pas créer de buffer supplémentaire, « récupérer » simplement celui qui correspond au bon **PageId**, après l'avoir rempli si besoin par un appel au **DiskManager,** via le pointeur ver le DiskManager stocké en variable membre.*

Attention aussi : c'est cette méthode (et non pas la suivante !) qui devra *s'occuper du remplacement du contenu d'un buffer si besoin ! C'est donc ici que vous allez devoir appliquer la politique de remplacement courante.*

- Une méthode
  void **FreePage** (*pageId*, *valdirty*) avec *pageId* un **PageId** et *valdirty* un entier ou booléen.

  Cette méthode devra décrémenter le pin_count et actualiser le flag dirty (et aussi potentiellement actualiser des informations concernant la politique de remplacement). Surtout ne faire aucun appel au **DiskManager** ici !

- Une méthode
  void **SetCurrentReplacementPolicy** (*policy*), avec *policy* une chaine de caractères ou tout autre représentation (enum, entier…) que vous avez choisi pour représenter les politiques à implementer.

  Cette méthode va changer la politique de remplacement courante, et aura la priorité par rapport à la politique spécifiée par la **DBConfig** passée au constructeur.

- Une méthode
  void **FlushBuffers**()

  Cette méthode s'occupe de :
  - l'écriture de toutes les pages dont le flag dirty = 1 sur disque (en utilisant le **DiskManager**)
  - la remise à 0 de tous les flags/informations et contenus des buffers. Après appel de cette méthode, le **BufferManager** repart avec des buffers où il n'y a aucun contenu de chargé (comme dans son état initial après appel du constructeur !)

B. Code : tests du BufferManager

Vous êtes libres de concevoir vous-mêmes les tests du **BufferManager**, en vous inspirant si vous le souhaitez des consignes pour le **DiskManager**. En particulier, nous vous conseillons de créer une classe **BufferManagerTests** similaire à **DiskManagerTests** et de tester avec des « petites » valeurs pour les paramètres (en particulier la taille d'une page et le nombre de buffers. N'oubliez pas de faire des tests qui déclenchent la politique de remplacement !

ATTENTION : les bugs dans le **BufferManager** sont fréquents, et difficiles à trouver lorsque vous serez très avancés dans le développent de votre application. Ces bugs ont tendance à causer toutes sortes de crash et comportement inattendus du SGBD, et donc à vous donner des migraines.
=> *Prenez vraiment le temps qu'il faut pour écrire des tests spécifiques au **BufferManager**, et assurez-vous qu'il marche bien avant d'attaquer les couches au-dessus !*