Bases de Données Avancées : TP2

Pendant ce TP, vous allez attaquer la première couche de votre SGBD : **le gestionnaire de l'espace disque**. Vous allez également travailler sur une petite classe qui centralisera les paramètres de votre SGBD.

Lorsque vous allez coder le gestionnaire disque, il faudra faire quelques petits raisonnements « par vous-même » (sans demander de l'aide auprès du chargé de TP) pour comprendre comment calculer certains éléments. Prenez donc le temps de réfléchir et essayez de bien comprendre comment un identifiant de page (PageId) est construit et comment les pages sont stockées sur disque !

A. Information : bien « paralléliser » votre travail

Même si vous n'êtes pas présents au TP en même temps, il est important de pouvoir travailler « en parallèle » avec son équipe, et donc de savoir distribuer les différentes tâches de programmation.

Pour cela, il faudra souvent que l'un de vous code « rapidement » des méthodes « vides » – qui puissent être par la suite appelées par le code de son collègue (car si ces méthodes n'existent pas, le code du collègue ne compilera pas...)

Un cas particulier : celui des tests. Nous vous conseillons vivement de développer en parallèle le code d'une classe et les tests associés à la classe ! C'est une distribution très naturelle du travail et cela permet de bien comprendre ce que la classe fait / doit faire !

B. Documentation : buffers et fichiers binaires

B1. Buffers / tableaux d'octets

Trouvez (pour le langage que vous avez choisi) une représentation convenable pour un buffer – autrement dit, pour un tableau (séquence contiguë) d'octets.

Des tableaux tout simples de char / unsigned char C ou byte Java conviennent sans problème, mais rien ne vous empêche d'utiliser d'autres structures si elles vous semblent plus conviviales.

En Java d'ailleurs, une classe qui convient très bien (et que nous vous conseillons !) est **ByteBuffer**. Jetez un coup d'œil à sa documentation !

B2. Fichiers binaires

Essayez de vous familiariser avec la lecture et l'écriture dans les fichiers binaires suivant le langage que vous avez choisi. En particulier, essayez de comprendre comment :

- ouvrir un fichier binaire en lecture et/ou écriture
- écrire et lire un buffer comme ci-dessus à un offset (une position) donné dans le fichier
- rajouter une séquence d'octets à la fin du fichier.

En C, vous pouvez par exemple jeter un coup d'œil à **fread/fwrite/fseek** etc ; en C++ à **fstream** ; en Java à **RandomAccessFile**.

C. Code : la classe DBParams

Nous allons introduire progressivement plusieurs « paramètres » de votre SGBD (vous avez déjà vu un premier tel paramètre : le chemin vers votre dossier DB!).
Nous allons regrouper ces paramètres dans une classe à part, appelée **DBParams**.

Créez la classe **DBParams** et rajoutez-y trois variables *publiques et statiques* :

- une variable **DBPath** qui correspond au chemin vers le dossier DB.
- une variable **SGBDPageSize** qui correspond à la taille d'une page.
- une variable **DMFileCount** qui correspond au nombre maximal de fichiers gérés par le Disk Manager (voir ci-dessous)

*Attribuez ensuite des valeurs à ces variables au tout début de la méthode **main** :*

- pour **DBPath**, *la valeur du premier argument de votre application*
- pour **SGBDPageSize**, la valeur *4096*
- pour **DMFileCount**, la valeur *4*.

En C vous pouvez faire une struct **DBParams** et rajouter une variable globale de type **DBParams.** Surtout, contrairement à Java, il ne faut pas rendre statiques les membres de la structure !

D. Code : gestion de l'espace disque

Votre SGBD *stockera toutes les données dans des fichiers* (→ fichiers « normaux », au sens OS).

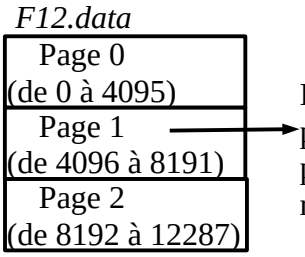
Ces fichiers s'appelleront *Fx.data* avec x entier >=0 l'identifiant du fichier : *F0.data*, *F1.data* etc. («data» est une extension que nous inventons pour ce TP :))

Il y aura au maximum **DMFileCount** fichiers qui stockeront ces données.
Tous ces fichiers *seront placés dans votre sous-répertoire DB*.

Pour lire et écrire dans ces fichiers, vous allez utiliser des *méthodes de lecture/écriture dans les fichiers binaires*.

Exemple :

le fichier **F12.data**
a 3 pages ;
la taille de ce fichier est
3 * pageSize = 3 * 4096 octets
= 12288 octets (de 0 à 12287)



Le PageId (voir ci-dessous) de cette page est (12,1). En regardant le premier élément du PageId on retrouve l'indice (12) du fichier.

C1. PageId

Chaque page dans chaque fichier sera identifiée par son PageId.
Le PageId est composé de deux parties :

- l'identifiant du fichier (un entier, le « x » dans Fx)
- l'indice de la page dans le fichier (0 pour la première page, 1 pour la deuxième etc.).

Étant donné un PageId, nous pouvons ainsi retrouver sans ambiguïté le fichier et l'emplacement de la page dans ce fichier, donc le contenu de la page.
Créez une classe **PageId** avec deux variables membres : FileIdx et PageIdx.

C2. DiskManager

Créez une classe **DiskManager** *qui comportera une seule et unique instance.*

Rajoutez-y (au moins) les méthodes suivantes, qui forment « l'API » du gestionnaire disque (ces méthodes seront appelées par les couches plus hautes!)

- *pageId* **AllocPage** (), avec *pageId* un **PageId**.
  Cette méthode doit allouer une page, c'est à dire réserver une nouvelle page à la demande d'une des couches au-dessus.

  L'algorithme pour l'allocation doit être le suivant :
  - Si une page désallouée précédemment (voir ci-dessous) est disponible, l'utiliser
  - Sinon, rajouter une page (c'est à dire, rajouter *pageSize* octets, avec une valeur quelconque, à la fin du fichier) dans le fichier de la plus petite taille disponible (les fichiers qui n'ont pas encore des pages rajoutées ont une taille 0, que vous les ayez créés ou pas, donc sont à considérer en priorité!)

  La méthode AllocPage retourne un PageId correspondant à la page nouvellement rajoutée !
  - *À vous de comprendre comment remplir ce PageId !*

- void **ReadPage** (*pageId*, *buff*) avec *pageId* un identifiant de page et *buff* un buffer (utiliser le type choisi : byte[], ByteBuffer....).
  Cette méthode doit remplir l'argument *buff* avec le contenu disque de la page identifiée par l'argument *pageId*. Il s'agit d'une page qui « existe déjà », **pas d'allocation dans cette méthode !**
  Attention : c'est l'appelant de cette méthode qui crée et fournit le buffer à remplir!

- void **WritePage** (*pageId, buff*) avec *pageId* un identifiant de page et *buff* un buffer (utiliser le type choisi : byte[], ByteBuffer.…)
  Cette méthode écrit le contenu de l'argument *buff* dans le fichier et à la position indiqués par l'argument *pageId.*

- void **DeallocPage** *(pageId), avec pageId un* ***PageId****.*
  Cette méthode doit désallouer une page, et la rajouter dans la liste des pages « disponibles ».

- int **GetCurrentCountAllocPages**()
  Cette méthode doit retourner le nombre courant de pages allouées auprès du **DiskManager.**
  Par exemple, après deux appels à **AllocPage** et un appel à **DeallocPage** elle doit retourner 1.

Attention : respectez impérativement la signature (arguments et type de retour) demandée pour chaque méthode !

D. Code : les tests du DiskManager

D'abord un conseil général : prenez le temps de bien tester chaque classe que vous codez, et en particulier les classes « bas-niveau » essentielles comme le **DiskManager** et le **BufferManager** (que nous verrons le TP suivant).
Un bug dans une de ces couches « bas-niveau » compromet en effet souvent tout le fonctionnement du SGBD, et sera plus difficile à trouver plus tard, quand votre projet aura acquis beaucoup plus de complexité !

Très important : le code qui correspond à vos tests fait partie de votre projet ! Il ne doit pas être effacé ou mis en commentaire, et la bonne approche c'est de refaire exécuter tous les tests d'une « brique » du SGBD à chaque fois que vous modifiez quelque chose dans la brique en question ! Vous allez ainsi détecter très vite les *régressions* (= « *ça marchait avant mais je viens d'y introduire un bug* » ;))

Vous allez apprendre plein de choses sur les tests (et en particulier les tests unitaires) dans votre cours de Programmation Avancée. Pour les tests de votre projet BDDA nous ne vous demandons pas de respecter un quelconque framework ou formalisme ; vous êtes libres de faire vos tests comme vous le voulez – juste ne les zappez pas ! :)

Ci-dessous quelques suggestions pour les tests du **DiskManager** :

- Pour vos tests, créez une classe **DiskManagerTests** qui a une méthode **main**. Rajoutez chaque test en tant que méthode sur la classe, par exemple **TestEcriturePage**, et appelez chaque telle méthode depuis la méthode **main** de **DiskManagerTests**. Vous ne « mélangerez » ainsi pas vos tests à votre SGBD (qui a sa propre méthode **main**).
- Dans vos tests, essayez d 'écrire puis lire du contenu dans les fichiers/pages, pour vérifier qu'à la lecture on retrouve bien les informations écrites précédemment !
- Pour faciliter les tests, attribuez (dans vos méthodes de test) pour la taille d'une page (donc le paramètre **SGBDPageSize**) une valeur plus « conviviale » que 4096 (par exemple 4, ou même 1!). Vous pourrez ainsi afficher et vérifier rapidement ce qui se trouve dans une page.
- Vous pouvez également faire varier **DMFileCount**, pour vérifier le bon usage de ce paramètre dans votre code !