Bases de Données Avancées : TP2

Pendant ce TP, vous allez attaquer la première couche de votre SGBD : **le gestionnaire de l'espace disque**.

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 et coder certains éléments. Prenez donc le temps de réfléchir et essayez notamment 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/C++ ou byte Java conviennent sans problème, mais rien ne vous empêche d'utiliser d'autres structures si elles vous semblent plus conviviales (par ailleurs, avec l'évolution des TPs, il sera potentiellement utile que vous puissiez « écrire » directement dans vos buffers des entiers, floats ou chaines de caractères….).

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 !

Pour Python, grâce au travail de Benoit Deveaux, une classe similaire est disponible sur Moodle.

Pour Rust, vous pouvez regarder https://crates.io/crates/bytebuffer .

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

C. Information : stockage des données binaires et fonctionnement du gestionnaire disque

Votre SGBD stockera toutes les données correspondant aux tables en format binaire dans des fichiers appellés *Fx.rsdb* avec x entier >=0 l'identifiant du fichier : *F0.rsdb* , *F1.rsdb* etc. («*rsdb*» est une extension que nous inventons pour ce TP :)).

L'ensemble de ces fichiers sera placé dans le sous-dossier **BinData** du dossier de stockage des données défini par le champs *dbpath* de l'instance de **DBConfig** utilisée par le SGBD.

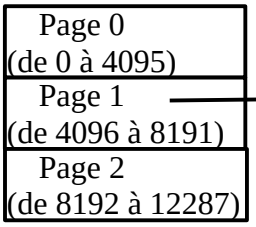
Toutes les allocations et lectures / écritures dans ces fichiers se feront *par pages.* Pour le fonctionnement du gestionnaire disque nous allons rajouter deux informations dans **DBConfig** (voir également les détails d'implémentation ci-dessous) :

- la taille d'une page (*pagesize*)
- la taille maximale d'un fichier (*dm_maxfilesize*).

Le gestionnaire disque gardera également en mémoire (et conservera d'une exécution à l'autre du SGBD) *une liste des pages « libres » dans les fichiers* (voir détails d'implémentation allocation / déallocation ci-dessous).

Suivant le choix d'identification des fichiers par leur indice (le « x » entier dans le nom), une page sera identifiée par un couple de valeurs (indice de fichier, indice de la page dans le fichier). L'exemple ci-dessous montre la structure d'un fichier pour des pages de taille 4096 octets, ainsi que la manière d'obtenir les identifiants des pages (PageId).

le fichier **F12.rsdb**
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.

D. Code : rajout d'informations dans DBConfig

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

- *pagesize*, correspondant à la taille d'une page
- *dm_maxfilesize*, correspondant à la taille maximale d'un fichier « rsdb » géré par le gestionnaire disque.

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 pour prendre en compte ces rajouts.

E. Code : PageId

Chaque page dans chaque fichier sera identifiée par son PageId, 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.

F. Code : DiskManager

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

- Un constructeur qui prend en argument une instance de **DBConfig** et qui stocke en variable membre une copie ou un pointeur / référence vers cette instance.

- Une méthode
  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 (liste des pages libres non-vide), l'utiliser
  - Sinon, rajouter une page (c'est à dire, rajouter *pagesize* octets (*pagesize* sera donné par l'instance de DBConfig passée au constructeur), à la fin d'un fichier qui n'a pas atteint sa taille maximale (*db_maxfilesize*). Si tous les fichiers ont atteint la taille maximale, le gestionnaire disque devra créer et remplir un nouveau fichier.

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

- Une méthode
  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* en copiant dans ce buffer le contenu disque de la page identifiée par l'argument *pageId*.

  Attention : c'est l'appelant de cette méthode qui crée et fournit le buffer à remplir! Il est erronné de déleguer la création du buffer au gestionnaire disque !

- Une méthode
  void **WritePage** (*pageId*, *buff*) avec *pageId* un identifiant de page et *buff* un buffer (utiliser le type choisi : byte[], ByteBuffer.…).

  Cette méthode écrit (copie) le contenu de l'argument *buff* dans le fichier et à la position indiqués par l'argument *pageId.*

- Une méthode
  void **DeallocPage** *(pageId),* avec *pageId un* ***PageId****.*

  Cette méthode doit désallouer une page, et la rajouter dans la liste des pages «libres».

- Une méthode
  void **SaveState**()

  Cette méthode devra sauvegarder dans un fichier au format de votre choix la liste des pages libres du gestionnaire disque. Le fichier s'appellera *dm.save* et sera placé à la racine du dossier *dbpath*.

- Une méthode
  void **LoadState**()

  Cette méthode devra charger la liste des pages libres depuis le fichier *dm.save* (voir ci-dessus).

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

Libre à vous de choisir les variables membres et méthodes supplémentaires que vous voulez rajouter dans le gestionnaire disque.

G. Code : les tests du DiskManager

Rappel : ne zappez pas les tests ! ;)

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

- Pour vos tests, si vous choisissez de contourner les infrastructures standard de tests unitaires, vous pouvez créer 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 aura 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, vous pouvez créer une configuration avec une taille de page de 4, voire 1 octet. Vous pourrez ainsi afficher et vérifier rapidement ce qui se trouve dans une page.
- N'oubliez pas de tester les méthodes Load et Save, pour vous assurer de la persistance des données du gestionnaire disque !