Pendant ce TP, vous allez démarrer le travail sur la couche Accès/Fichiers, plus particulièrement sur la classe (essentielle à notre SGBD) **HeapFile**.
Cette classe nous permettra de lire et écrire des records, donc de traiter des commandes de type **insert** et **select**.
Nous allons commencer par coder quelques méthodes « bas-niveau » : des briques qui seront ensuite utilisées par les méthodes d'insertion et de sélection.

Prenez le temps de *lire très attentivement* la partie « Information » ci-dessous, pour pouvoir bien coder les différents éléments.
Il conviendra également de revenir sur ces descriptions, si besoin, lors des TPs suivants !

A. Information : Heap Files. A LIRE ATTENTIVEMENT !!!

Chaque Heap File correspondra à une relation, donc (cf TP2) il y aura un fichier disque (Data_x.rf) associé.

Nous allons utiliser des Heap Files avec une gestion des pages de type *Page Directory* et un stockage de type *unpacked/bitmap pour des records de taille fixe*.
(→ *si cela vous semble incompréhensible, direction les diapos du Cours3 sur Moodle*).

Nous allons de plus faire un ensemble de (grandes) simplifications / ajustements sur le format de la Header Page, ainsi que des pages de données, comme décrit ci-dessous.

A1. La Header Page

La Header Page sera toujours la première page du fichier disque correspondant au Heap File.
C'est donc *la page d'indice 0*.
Cela veut dire que si on connaît l'indice (le « x » dans Data_x.rf) de ce fichier, on connaît le PageId de la Header Page.

Chaque objet Heap File aura accès à l'indice du fichier correspondant.
Pour récupérer la Header Page via le BufferManager, *nous allons donc toujours utiliser un PageId composé de l'indice du fichier et de la valeur 0*.

Nous allons ensuite supposer qu'une seule page suffit pour stocker tout le Page Directory (pas de liste chaînée).

Le contenu de la Header Page sera le suivant :

- au tout début : un entier *N*, correspondant au nombre de pages de données dans le fichier
- ensuite *N* entiers correspondant au nombre de slots (cases) restant disponibles sur chacune des *N* pages de données

Les pages de données sont l'une à la suite de l'autre, en commençant par l'indice 1 (donc la deuxième page du fichier – pourquoi ?).

A2. Les pages de données

Plutôt que d'utiliser une bitmap, pour vous simplifier le travail, nous allons utiliser une « bytemap » :
Pour chaque slot de la page nous aurons *un octet qui vaut 0 ou 1*, en fonction du statut (libre / occupé) du slot.

Soit *slotCount* le nombre de cases sur une page de données.
Une page de données contiendra donc, au début, *slotCount* octets pour la « bytemap », suivis des *slotCount* cases (pleines ou vides), chacune de la taille d'un record.

Prenant cela en compte, le nombre de cases sur une page est calculable si on connaît la taille d'un record et la taille d'une page. À vous de faire ce calcul !!

B. Code : recordSize, slotCount et fileIdx dans la classe RelDef

Rajoutez, sur la classe **RelDef**, trois variables membres (entiers) :

- **fileIdx**, qui correspond à l'indice du fichier disque qui stocke la relation
- **recordSize** qui représente la taille d'un record
- **slotCount**, qui représente le nombre de cases (slots) sur une page

pour la relation en question.

Modifiez le **constructeur** de **RelDef** pour prendre en argument ces trois éléments.

La valeur de **recordSize** correspond à la taille en octets d'un record quand on l' « écrit sur une page ». Cet entier sera donc calculé *en fonction du type des colonnes de la relation* :

- Pour les gens qui codent en Java : vous allez pouvoir utiliser *4 pour les int*, *4 pour les float*, et *x*2 pour les stringx* (car attention, le type char en Java occupe deux octets).
  Exemple : si une relation a 4 colonnes de type string7, int, int et string3 alors la taille d'un record = 14+4+4+6 = 28.

La valeur de **slotCount** est calculable en fonction de **pageSize** (constante) et du **recordSize** ci-dessus. Voir aussi le point A2.

Rajoutez le calcul de **recordSize** et **slotCount** dans la méthode **CreateRelation** du **DBManager,** *avant la création de la RelDef***.** Le constructeur de la RelDef utilisera donc ces valeurs, ainsi que *la valeur du compteur courant de relations de la DBDef* comme valeur pour l'argument *fileIdx*.

C. Code : Persistance de la DBDef / catalogue

D'une exécution à l'autre de votre application, il y aura besoin de stocker/persister les informations contenues dans la **DBDef**.
Pour ce faire, nous allons « sauvegarder » ces informations dans un fichier sur disque à l'arrêt de l'application ; puis, au démarrage, nous allons « remplir » la **DBDef** avec les informations contenues dans ce fichier.

C1. Ecriture.

Rajoutez, dans la méthode **Finish** de la classe **DBDef**, du code qui sauvegarde les informations de la **DBDef** dans un fichier nommé *Catalog.def.* Ce fichier sera *placé dans le dossier DB.*

Vous pouvez écrire les informations dans le format de votre choix (fichier texte, ou bien utiliser la *sérialisation Java* et **ObjectOutputStream**…).
L'essentiel c'est que *tout ce qui est contenu par la DBDef* (nombre de relations, et pour chaque relation les informations de schéma, etc.) *soit bien sauvegardé*.

C2. Lecture

Rajoutez, dans la méthode **Init** de la classe **DBDef**, du code qui lit les informations du fichier *Catalog.def* et remplit la **DBDef** avec ces informations.
Cas à part : si pas de fichier *Catalog.def* présent dans le dossier DB, alors pas besoin de remplir quoi que ce soit !

D. Code : La classe Rid

Rajoutez une classe **Rid** qui correspond au Rid (Record Id, donc identifiant) d'un Record.
Votre classe contiendra deux variables membres :

- *pageId*, un **PageId** qui indique la page à laquelle appartient le Record
- *slotIdx*, un entier qui est l'indice de la case où le Record est stocké.

E. Code : Creation du fichier disque pour un Heap File.

Rajoutez une classe **HeapFile** avec une seule variable membre : *relDef*, une **RelDef**.

Rajoutez sur cette classe une méthode
createNewOnDisk ()

Cette méthode devra gérer :

- La création du fichier disque correspondant au Heap File.
  Vous allez utiliser pour cela la méthode **CreateFile** du **DiskManager**.
  L'indice de fichier à utiliser est donné par *relDef***.**
- Le rajout d'une Header Page « vide » à ce fichier.
  Pour cela vous allez d'abord utiliser **AddPage** (du **DiskManager**),
  puis récupérer la page nouvellement rajoutée via le **BufferManager,**
  puis écrire pageSize octets de valeur 0 dedans,
  et enfin la libérer auprès du **BufferManager** (*avec le bon flag dirty*).

F. Code : Rajout d'une page de données au Heap File.

Rajoutez, sur la classe **HeapFile**, une méthode
pageId **addDataPage()**
avec *pageId* un **PageId**.

Cette méthode devra rajouter une page de données et retourner le PageId de cette page.
Pour cela, elle devra :

- rajouter une page au fichier disque correspondant (via le **DiskManager**)
- puis *actualiser les informations de la Header Page.*

ATTENTION : pour écrire dans la Header Page, il faut toujours la récupérer via le BufferManager, puis la libérer avec dirty = 1.
N'oubliez pas, aussi, que la Header Page est la première page dans le fichier disque (donc indice 0).

G. Code : Trouver une page libre dans un Heap File.

Rajoutez, sur la classe **HeapFile**, une méthode
pageId **getFreeDataPageId()**.
avec *pageId* un **PageId**.

Cette méthode doit retourner le PageId d'une page de données qui a encore des cases libres.

Pour cela, il faudra lire les informations de la Header Page.
N'oubliez pas d'obtenir la Header Page via le **BufferManager**, et de la libérer après lecture.

Si aucune des pages de données ne contient des cases libres, la méthode doit retourner ***null***.

H. Code : Ecrire un Record dans une page de données

Rajoutez, sur la classe **HeapFile**, une méthode
rid **writeRecordToDataPage** (*record, pageId*)
avec *record* un **Record**, *pageId* un **PageId** et *rid* un **Rid**.

Cette méthode doit écrire *record* dans la page de données identifiée par *pageId, et renvoyer son Rid (voir point D ci-dessus)!*

Nous supposons que la page dispose d'au moins une case disponible (pas besoin de vérifier!)

Pour écrire le Record, il faut récupérer la page de données via le Buffer Manager.
Ensuite, il faut trouver une case disponible en consultant la bytemap.
Une fois la page de données actualisée, n'oubliez pas de la libérer auprès du BufferManager (quel flag *dirty*?)

Enfin, n'oubliez pas d'actualiser la Header Page (car le nombre de cases disponibles de la page de données a changé!)

Cette méthode devra utiliser une des méthodes de la classe Record que vous avez codée lors du TP précédent.

I. Code : Enumérer les Records d'une page de données

Rajoutez, sur la classe **HeapFile**, une méthode
listeDeRecords **getRecordsInDataPage** (*pageId*)
avec *pageId* un **PageId** et *listeDeRecords* une liste (ou un tableau) de **Record**.

Cette méthode doit retourner la liste des records stockés dans la page identifiée par *pageId*.
Utilisez une des méthodes de la classe Record que vous avez codée lors du TP précédent.