Pendant ce TP, vous allez démarrer en douceur votre projet : vous allez choisir votre équipe et la manière de vous organiser pour le travail collaboratif, lire beaucoup de consignes et un peu de documentation, travailler sur la structure globale de votre application… et même coder un peu. Pour le code, n'oubliez pas de vous organiser (distribuer les tâches) de manière à travailler en parallèle !

Dans ce descriptif TP ainsi que les suivants, les « choses à faire » sont classées en trois catégories : **Information** (à lire attentivement – pour agir conformément! ;)), **Documentation** (chercher comment faire pour la suite ; pas de code dans l'immédiat) et **Code** (...).

A. Information : but et fonctionnalités de votre projet

Comme vous devez déjà le savoir, votre projet concernera l'implémentation d'un « MiniSGBDR », autrement dit un SGBDR très (TRES !) simplifié.

Votre SGBD comprendra une seule base de données et un seul utilisateur ! Pas de gestion de la concurrence, pas de transactions, pas de droits d'accès, pas de crash recovery…

Les commandes que votre SGBD devra traiter seront dans un langage beaucoup plus simple que SQL, tout en correspondant à des « vraies commandes SQL » : insertion, sélection, etc.
Vous allez progressivement travailler sur ces commandes au fil des TPs, après avoir codé les couches « bas-niveau » du SGBD.
Tout ceci va suivre, autant que possible, le rythme du cours (nous allons incorporer progressivement des éléments vus en cours).

B. Information : dossiers, fichiers etc.

Vous allez placer votre projet dans un répertoire Projet_Nom1_Nom2… (les noms des membres de l'équipe, donc).

Votre répertoire devra contenir **un sous-répertoire Code** (avec le code;)) et **un autre sous-répertoire DB** (avec le contenu de la base de données), ainsi qu'**un ou plusieurs** (si vous souhaitez tester sur plusieurs OS) **scripts** pour lancer l'exécution de votre SGBD.

Testez toujours le lancement de votre projet avec le(s) script(s), même si pour le debug vous préférez lancer depuis un IDE (Eclipse…).
Si vous testez votre code sur des OS différents, attention aux différences possibles entre les séparateurs ('/', '\') dans la gestion des chemins fichiers.
Libre à vous de créer, si vous en voyez l'utilité, des sous-répertoires du répertoire Code.

C. Information : application console et boucle de traitement des commandes

Votre projet sera une application console, sans interface graphique.

Le déroulement typique d'une utilisation de votre application sera : lancement via le script, puis série de commandes qu'on tape dans la console et auxquelles l'application réagit, ceci jusqu'à ce qu'on lui donne la commande **exit**.

Si vous souhaitez (mais ce n'est pas du tout obligatoire), vous pouvez afficher un « command prompt » (par exemple la phrase : « tapez votre commande (svp)») lorsque votre application est en attente d'une commande.

D. Information : terminologie POO

Les descriptifs TP feront toujours référence à des classes / méthodes.
Si vous utilisez un langage qui n'est pas orienté objet, il faut adapter ces descriptifs au langage que vous utilisez: par exemple, en C, vous allez créer une struct à la place de la classe, et des fonctions qui prennent en argument supplémentaire un pointeur sur la struct à la place des méthodes…
Rappel : utilisez pour votre projet un langage que vous maîtrisez bien !

E. Information : singletons – instances uniques

Plusieurs classes « essentielles » de votre application devront comporter une seule et unique instance.
Une manière de s'assurer de cela c'est de leur faire correspondre des variables statiques (globales pour C).
Pour faire joli, jetez un coup d'oeil au design pattern Singleton (mais un design moins soigné sera tout aussi efficace, à partir du moment où vous garantissez l'unicité !)

F. Information : méthodes Init, Finish

Plutôt que de rajouter des actions type « logique DB » dans les constructeurs (destructeurs), on préférera souvent, au cours de ce projet, le rajout de deux méthodes :
- **Init**, qui fera le nécessaire pour l'initialisation d'une instance.
- **Finish**, qui elle s'occupera « du ménage »

Les deux méthodes seront de préférence sans arguments et sans argument de retour (void).

G. Documentation : buffers et fichiers binaires

G1. Buffers / séquences d'octets

Trouvez (pour le langage que vous avez choisi) une représentation convenable pour un buffer - 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 est **ByteBuffer**.

G2. Fichiers binaires

Essayez de vous familiariser avec la lecture et l'écriture dans les fichiers binaires suivant le langage de votre choix. 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 0 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**.

H. Code : début du DBManager = point d'entrée de la logique SGBD

Vous allez commencer à coder la classe qui fonctionne comme le « point d'entrée » de la logique spécifique SGBD.
Cette classe s'appellera **DBManager** et *on s'assurera qu'il en existe une seule et unique instance* (que nous appellerons *le* DBManager).

Rajoutez-y les méthodes **Init** et **Finish** et une méthode **ProcessCommand,** qui prend en argument une chaîne de caractères qui correspond à une commande.

I. Code : la boucle de traitement des commandes (main)

Rajoutez, dans la méthode **main** de votre application :

- la création (si besoin) de l'instance de **DBManager**
- l'appel de sa méthode **Init**
- la boucle de gestion des commandes.

Cette boucle prendra des commandes utilisateur (chaînes de caractères) et :

- si la commande est **exit**, alors il y a appel de **Finish** sur le **DBManager**, puis sortie de la boucle
- sinon, la commande est simplement passée au **DBManager**, via **ProcessCommand** (et la boucle continue).

J. Code : informations de schéma

J1. La classe RelDef

Créez une classe appelée **RelDef** (Rel=relation ; Def=définition) qui garde les informations « de schéma » d'une relation.
Cette classe devrait avoir comme membres :

- le nom de la relation (chaîne de caractères)
- le nombre de colonnes (entier)
- les types des colonnes (liste ou tableau de chaînes de caractères)

Pour les types de colonnes, on se restreindra aux :

- entiers (notés « **int** »)
- float (notés « **float** »)
- chaînes de caractères de taille fixe T (notées « **stringT** » ; par exemple pour chaîne de taille 3 le type string3).

On suppose que pour une relation à N colonnes, les colonnes sont nommées C1,…,CN. *Il n'y a donc pas besoin de stocker les noms des colonnes.*

J2. La classe DBDef

Créez une classe appelée **DBDef** qui contiendra les informations de schéma pour l'ensemble de votre base de données. *On s'assurera qu'il existe une seule et unique instance de cette classe dans l'application* (instance appelée *le* DBDef).

Les variables membres de votre classe seront :

- une liste ou tableau de **RelDef**
- un compteur de relations (entier)

Rajoutez à la classe les méthodes **Init** et **Finish**, ainsi qu'une méthode **AddRelation**, qui prend en argument une **RelDef** et qui la rajoute à la liste et actualise le compteur.

J3. Utilisation du DBDef dans le DBManager

Rajoutez, dans le **DBManager** :

- un appel à la méthode **Init** du **DBDef** dans la méthode **Init** du **DBManager**
- un appel à la méthode **Finish** du **DBDef** dans la méthode **Finish** du **DBManager**
- une méthode **CreateRelation** qui prend en argument le nom d'une relation, le nombre et les types de ses colonnes. Cette méthode créera une RelDef conformément aux arguments et la rajoutera au DBDef.

K. Code : la commande create

Rajoutez dans votre application la gestion de la commande **create**, qui crée une relation.

Le format de cette commande est le suivant :

create NomRelation NbCol TypeColl[1] TypeCol[2] ... TypeCol[NbCol]

Exemple:

create R 3 int float string10

Vous devrez pour cela rajouter du code au niveau de **ProcessCommand** dans le **DBManager**, et utiliser les éléments que vous venez de coder.