Bases de Données Avancées : TP1

Pendant ce TP, vous allez démarrer en douceur votre projet : vous allez lire beaucoup de consignes, définir la structure globale de votre application… et coder un peu.

Pour le code, n'oubliez pas de vous organiser (distribuer les tâches) de manière à travailler en parallèle avec votre binôme (même s'il n'est pas présent dans la salle TP)! Si vous rencontrez des difficultés pour paralléliser, n'hésitez pas à demander conseil à votre chargé de TP !

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 (Système de Gestion de Bases de Données Relationnelles) 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.

Par exemple, nous allons traiter des commandes de sélection très simples de type :
SELECT nomRelation WHERE nomColonne=valeurCible
qui demandent d'afficher tous les tuples de **nomRelation** dont la valeur sur la colonne **nomColonne** est **valeurCible**.

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).

Ayez toujours en tête le « but final » de votre SGBD : celui de savoir gérer les commandes de création de tables, sélection, jointure, etc. Il est facile des fois de s'y perdre lorsqu'on travaille sur les couches bas-niveau !

B. Information : structure de votre projet, dossiers, fichiers, scripts etc.

Vous allez placer votre projet dans un dossier Projet_BDDA_Nom1_Nom2… (les noms des membres de l'équipe, donc ; pas besoin de rajouter les prénoms).

Votre dossier devra contenir **un sous-dossier Code** (avec le code;)) et **un autre sous-dossier DB** (avec le contenu de la base de données), ainsi qu'**un ou plusieurs** (si vous souhaitez tester sur plusieurs systèmes d'exploitation) **scripts** pour l'exécution de votre SGBD. Libre à vous de créer, si vous en voyez l'utilité, des sous-dossiers du dossier Code.

Attention : il est essentiel de respecter la structure demandée (et les précisions ci-dessous), sous peine d'être pénalisés dans votre note finale de projet !

B1. Les projets Eclipse

Si vous faites du Java avec Eclipse, vous devez impérativement :

- appeler votre projet Eclipse **MiniSGBD_Nom1_Nom2…** (les noms des membres de l'équipe, donc ; pas besoin de rajouter les prénoms).
- placer le dossier de votre projet Eclipse (qui a le même nom que le projet Eclipse lui même) directement dans le dossier Code.

Si vous avez besoin d'aide, sollicitez votre chargé de TP ! Si vous vous êtes trompés dans le nommage et / ou le placement, il vaut mieux tout effacer et reprendre de 0 !

B2. Les scripts

Lors de l'évaluation de votre projet, votre application sera lancée à partir d'un script de lancement que vous aurez placé dans le dossier principal de votre projet. *Pas de lancement depuis Eclipse et autre IDE !*

Pour ceux que le mot « script » effraie : à l'intérieur de ces fameux « scripts », vous allez simplement retrouver des commandes de compilation et lancement de votre application !
Pour ceux qui font du Java, ce sera typiquement l'invocation de « java » et « javac ». On en parle dans votre cours de Programmation Avancée (si, si :)). Vous pouvez également regarder par ici : http://lampwww.epfl.ch/teaching/archive/compilation/html/compiling_java_programs.html.

Il est impératif que vous ayez un script qui fonctionne à la fin de ce TP. Votre chargé de TP est là pour vous aider si besoin !

C. 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 !

D. Information : application console, commandes, argument à l'exécution

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** (tout en majuscules).

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.

Votre application (et donc la fonction/méthode *main*) prend un seul argument : le chemin vers votre (sous)dossier DB.
Dans les scripts, vous devez faire figurer un **chemin relatif.** Vous pouvez, si vous le souhaitez, mettre un chemin absolu quand vous testez depuis Eclipse ou un autre IDE. Si vous avez des soucis pour spécifier les arguments dans votre IDE : votre chargé de TP vous aidera ! :)

E. Documentation : 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 et propre » :) dans le code Java, il est préférable d'utiliser le « design pattern » (patron de conception) **Singleton**; vous pouvez voir ce que c'est par exemple ici : https://www.codeflow.site/fr/article/java-singleton. Vous allez apprendre plus sur les design patterns vers la fin de votre cours de Programmation Avancée !

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. Code : classe Main

Rajoutez dans votre application une classe **Main** qui contient la méthode **main**.

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

Vous allez commencer à coder la classe qui sera 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**).

→ *Pour ce faire, implémentez le « **getInstance** » typique du Singleton. À chaque fois que vous aurez besoin d'accéder au **DBManager**, il faudra ainsi appeler cette méthode statique (**DBManager.getInstance()**) qui vous rendra l'instance **DBManager** à utiliser.*

Rajoutez sur la classe DBManager les méthodes **Init** et **Finish** et une méthode **ProcessCommand,** qui prend en argument une chaîne de caractères qui correspond à une commande.

Attention : aucune de ces méthodes ne doit être statique !!!

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

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

- l'appel de la méthode **Init** du **DBManager**
- 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 (donc la chaîne de caractères) est simplement passée au **DBManager**, via **ProcessCommand** (et la boucle continue).

Vous avez un exemple de boucle de traitement des commandes dans le corrigé du TP0.

J. Code : informations de schéma

J1. La classe RelationInfo

Créez une classe appelée **RelationInfo** qui gardera (entre autres) 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 noms des colonnes (liste ou tableau de chaînes de caractères)
- les types des colonnes (liste ou tableau de chaînes de caractères)

Attention : 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).

*Si vous le souhaitez, vous pouvez créer une classe supplémentaire appelée par exemple **ColInfo**, et mettre dans cette classe le nom et le type de la colonne ; puis stocker, dans la classe **RelationInfo**, une liste ou un tableau de **ColInfo**.*

J2. La classe DBInfo

Créez une classe appelée **DBInfo** qui contiendra (entre autres) 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 dans la suite des TPs *le* DBInfo).

Les variables membres de votre classe seront :

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

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

Attention : aucune de ces méthodes ne doit être statique !

J3. Utilisation du DBInfo dans le DBManager

Rajoutez, dans la classe **DBManager** :

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

K. Code : début de la commande CREATEREL

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

Le format de cette commande est le suivant :

CREATEREL NomRelation NomCol[1]:TypeColl[1] NomCol[2]:TypeCol[2] … NomCol[NbCol]:TypeCol[NbCol]

Exemple:

CREATEREL R C1:int C2:float C3:string10

Attention :

- pour chaque colonne, son nom et son type sont séparés par un « deux points ». il n'y a pas d'espace avant/après le «deux points» !
- le nom de la commande est en majuscules !

Pour l'instant, l'effet de cette commande sera simplement l'appel de la méthode **CreateRelation** du **DBManager** !

Pour avoir la gestion « d'un bout à l'autre » de cette commande, vous devrez donc rajouter du code au niveau de **ProcessCommand** dans le **DBManager** (« parsing » de la commande en utilisant par exemple la méthode **split** de la classe **String**, détection du mot clé **CREATEREL**, etc., pour finir sur un appel de la methode **CreateRelation** avec les bons arguments. Pour vérifier que tout va bien, vous pouvez afficher dans **CreateRelation** un joli « message de debug » indiquant par exemple le contenu des arguments ! ;)).

La gestion de la commande **CREATEREL** sera « enrichie » au cours des TPs suivants !

CHECKLIST A LA FIN DE CE TP :

- **Assurez-vous d'avoir bien parcouru et compris l'ensemble de ce (long) descriptif TP ! Soyez rassurés, les prochains TPs seront moins longs ! ;)**
- **Vérifiez que vous avez bien créé la structure de votre projet avec les dossiers et sous-dossiers demandés !**
- **Vérifiez que votre application se lance bien depuis le script !**
- **Vérifiez le fonctionnement des commandes CREATEREL et EXIT!**
- **Faites vos retours en direct auprès de votre chargé de TP ou par mail auprès de la chargée de cours ; dites-nous ce qui est clair – et ce qui l'est pas, quels sont les points qui vous paraissent difficiles, etc.**