Bases de Données Avancées : TP6

Avec ce TP, vous allez **enfin** commencer à coder la gestion des commandes type SQL ! \o/

A. Information : gestion de plusieurs bases de données. A LIRE ATTENTIVEMENT !!!

Votre SGBD devra gérer plusieurs bases de données, qui contiendront chacune des tables (il est possible aussi d'avoir une base de données vide).

On devra pouvoir créer et supprimer de telles base de données.
A l'intérieur de chaque base de données, on pourra rajouter et supprimer des tables.

La création et la suppression des tables se fera toujours dans la base de données active couramment.
L'utilisateur doit pouvoir à tout moment choisir une base de données active.

Ci-dessous vous trouverez la liste de commandes concernant les actions de création / suppression de base de données et de tables, ainsi que le choix de la base de données active.

Notez que lors de l'évaluation de votre projet, les scénarios testés contiendront toujours des commandes correctes et valides (pas d'erreurs dans la syntaxe, pas de demande de suppression d'une table inexistante ou de création d'une table « doublon » etc).
Si vous souhaitez enrichir votre code pour prendre en compte les cas d'erreur (qui peuvent être nombreux…) cela sera apprécié, mais reste toutefois un travail *optionnel, non-obligatoire.*

A1. Création d'une base de données

La commande de création d'une base de données a la forme suivante :

CREATE DATABASE NomBDD

avec **NomBDD** le nom de la base de données à créer. Ce nom contient *uniquement des lettres (sans accent) et des chiffres.*

A2. Choix de la base de données courante

La commande de choix d'une base de données courante a la forme suivante :

SET DATABASE NomBDD

avec **NomBDD** le nom de la base de données à activer (on suppose que cette base de données existe). Attention, la création d'une base de données n'implique aucunement sa sélection en bdd active !

A3. Création d'une table dans la base de données courante

La commande de création d'une table (relation) dans la base de données courante a la forme suivante :

CREATE TABLE NomTable (NomCol_1:TypeCol_1,NomCol_2:TypeCol_2, … NomCol_NbCol:TypeCol_NbCol)

Exemple:
CREATE TABLE R (X:INT,C2:REAL,BLA:CHAR(10))

Commentaires importants :

- 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» !
- il y a un seul espace séparateur avant et après le nom de la relation, et pas d'espace à partir de la parenthèse ouverte
- si vous souhaitez gérer un nombre d'espaces arbitraire entre les éléments de la commande, c'est possible et c'est tout à votre gloire et honneur ;)
- on suppose qu'une base de données est active lors de l'appel de cette commande
- plusieurs tables avec le même nom peuvent exister dans des bases de données différenters !

A4. Suppression d'une table dans la base de données courante
La commande de suppression d'une table (existante!) dans la base de données courante a la forme suivante :

DROP TABLE NomTable

A5. Affichage de toutes les tables dans la base de données courante
La commande d'affichage de toutes les tables (relations) dans la base de données courante a la forme suivante :

LIST TABLES

Cette commande affichera chaque table sur une ligne séparée. Pour chaque table, on affichera son nom ainsi que les noms et les types de ses colonnes, dans une format similaire à celui de la commande **CREATE TABLE.**

A6. Suppression de toutes les tables dans la base de données courante
La commande de suppression de toutes les tables (relations) dans la base de données courante a la forme suivante :

DROP TABLES

A7. Suppression de toutes les bases de donnéers existantes
La commande de suppression de toutes les bases de données a la forme suivante :

DROP DATABASES

Attention : une fois cette commande appelée, il n'y a plus de base de données active couramment !

A8. Affichage de toutes les bases de données existantes
La commande d'affichage de toutes les bases de données a la forme suivante :

LIST DATABASES

A9. Suppression d'une base de données
La commande de suppression d'une base de données a la forme suivante :

DROP DATABASE NomBDD

Attention : si la base de données supprimée est la base de données active, il n'y aura plus de base de données active.

B. Code : La classe DBManager et ses tests

Créez une classe **DBManager** dont les variables membres devront pouvoir assurer l'implementation des méthodes décrites ci-dessous.

En particulier, votre classe **DBManager** devra contenir une/des structure(s) pour stocker et rechercher (efficacement) toutes les bases de données et les relations à l'intérieur de chaque telle base de données. Pour les relations, vous allez utiliser la classe **Relation**, codée lors des TPs précédents.

Jetez un coup d'oeil aux structures de type « dictionnaire » / « map ». Vous pouvez aussi, si vous le souhaitez, créer des classes intermédiaires, par exemple une classe **Database** correspondant à une base de données.

La classe **DBManager** devra également stocker la base de données couramment active.

Implémentez, sur votre classe **DBManager**, les méthodes suivantes :

- Un constructeur qui prend en argument une instance de **DBConfig.**
- Une méthode
  void **CreateDatabase** (*nomBdd*), avec *nomBdd* une chaine de caractères correspondant au nom de la base de données à créer.
- Une méthode
  void **SetCurrentDatabase** (*nomBdd*), avec *nomBdd* une chaine de caractères correspondant au nom de la base de données à activer.
- Une méthode
  void **AddTableToCurrentDatabase** (*tab*), avec *tab* une instance de la classe **Relation**
- Une méthode
  tab **GetTableFromCurrentDatabase** (*nomTable*), avec *tab* une instance de la classe **Relation** et *nomTable* une chaine de caractères, qui retourne l'instance de **Relation** correspondant à *nomTable*.
- Une méthode
  void **RemoveTableFromCurrentDatabase** (*nomTable*), avec *nomTable* une chaine de caractères qui désigne le nom de la table à supprimer.
- Une méthode
  void **RemoveDatabase** (*nomBdd*), avec *nomBdd* une chaine de caractères qui désigne le nom de la base de données à supprimer.
- Une méthode
  void **RemoveTablesFromCurrentDatabase** ()
- Une méthode
  void **RemoveDatabases** ()
- Une méthode
  void **ListDatabases** (), qui affiche les noms de toutes les bases de données existantes, une par ligne.

- Une méthode
  void **ListTablesInCurrentDatabase** (), qui affiche les noms et les schémas (noms et types des colonnes) de toutes les tables de la base de données couramment active, une table par ligne, dans un format similaire à celui de la commande de création de table décrite ci-dessus.

- Une méthode
  void **SaveState**()

  Cette méthode devra sauvegarder dans un format de votre choix toutes les informations correspondant aux bases de données et aux tables appartenant à chaque base de données.

  Un exemple de format possible est un fichier unique, appelée *databases.save*, placé à la racine du dossier *dbpath* (rappel, ce dossier est donné par l'instance de **DBConfig** passée en argument du constructeur). Vous pouvez également créer un sous-dossier , et/ou des fichiers séparés pour chaque base de données.

  ATTENTION : pour sauvegarder correctement les informations correspondant à une relation, n'oubliez pas de sauvegarder l'identifiant de sa *header page* (variable membre *headerPageId* de la classe **Relation**).

- Une méthode
  void **LoadState**()

  Cette méthode devra charger les informations correspondant aux bases de données, suivant le format de sauvegarde que vous avez choisi.

La méthode *SaveState* sera typiquement appelée à l'arrêt de l'exécution du SGBD. Elle permettra de sauvegarder les informations concernant les bases de données et les tables existantes. Ainsi, lors d'un prochain démarrage de votre application, ces informations pourront être récupérées via *LoadState,* et le utilisateurs pourront continuer d'utiliser les tables créées précédemment.

En complément de l'implémentation de toutes les méthodes décrites ci-dessus, prenez le temps d'écrire des tests qui couvrent toutes les fonctionnalité de votre classe **DBManager**.

C. Code : La classe SGDB et ses tests

La classe principale de votre SGBD sera appelée … **SGBD** (eh oui).
Cette classe contiendra également la méthode (statique) *main* qui sera le point d'entrée de votre application.

Créez la classe **SGBD** et rajoutez-y les méthodes suivantes :

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

  Le constructeur devra créer et stocker en variables membres une instance de **DiskManager,** une instance de **BufferManager** et une instance de **DBManager**.

  Il devra également appeler *loadState* sur l'instance de **DiskManager** et sur celle de **DBManager.** Cela lui permettra de « récupérer les données existantes », créées lors des utilisations précédentes de votre application.

- Une méthode
  void **Run**()
  qui sera détaillée ci-après.

- Des méthodes
  void **Process*X*Command** (*texteCommande*), une pour chauqe commande traitée, où le « X » correspondra à la commande (par exemple, **ProcessSetDatabaseCommand)** et où *texteCommande* représente la chaine de caractères correspondant à la commande.
  Plus de détails sur la gestion de certaines commandes seront présentés ci-après.

- Une méthode statique *main* (« la » méthode *main* de votre application). Votre application prendra en seul et unique argument (récupérable donc via la méthode main) une chaine de caractères correspondant au chemin vers le fichier de configuration.

  La méthode *main* aura comme missions de :
  - Construire l'objet **DBConfig** à partir du chemin
  - Créer une instance de la classe **SGBD** en lui passant en argument l'objet **DBConfig**
  - Appeler, sur l'instance de la classe **SGBD** construite, la méthode *Run*.

C1. La méthode Run et la gestion des commandes

La méthode *Run* devra contenir une *boucle de traitement des commandes*, qui va demander (en présentant potentiellement un « prompt » comme « ? » ou « veuillez svp svp taper une commande ») à l'utilisateur de taper une commande, puis va exécuter cette commande.

La boucle sera interrompue par la saisie, par l'utilisateur, de la commande **QUIT**. Il faudra donc coder une méthode **ProcessQuitCommand** sur la classe **SGBD**.

A l'intérieur de la boucle, il y aura une première étape d'analyse / *parsing* d'une commande, essentiellement pour savoir de quel type de commande il s'agit. Par la suite, le traitement de la commande (y inclus l'analyse / *parsing* complémentaire) sera délégué à la méthode **ProcessXCommand** qui convient.

Dans la description ci-dessus de ces méthodes il est indiqué de mettre en argument la chaine de caractères correspondant à la commande. Toutefois, si vous le souhaitez, vous pouvez passer en argument le résultat du *parsing* déjà effectué (par exemple, si vous avez segmenté la chaine de caractères en prenant les espaces comme séparateurs…).

Quelques précisions supplémentaires concernant la gestion de certaines commandes présentées dans ce TP :

- La commande **QUIT**, qui va terminer l'exécution courante du SGBD, devra s'assurer de la sauvegarde de toutes les informations utiles, via des appels aux méthodes S*aveState et FlushBuffers.*
- La commande **CREATE TABLE** devra créer un objet de type **Relation,** en n'oubliant pas de créer et d'initialiser sa *header page*. Cet objet devra ensuite être rajouté à la base de données courante.
- La commande **DROP TABLE** devra non seulement supprimer l'instance de **Relation** correspondante dans le **DBManager**, mais aussi désallouer auprès du **DiskManager** toutes les pages qui appartiennent à cette relation. Même principe pour les commandes **DROP TABLES** et **DROP DATABASES**.

C2. Un exemple de scénario de test de votre application

Quand vous avez fini d'implémenter la gestion des commandes, vous pouvez tester votre application sur le scénario suivant :

```
CREATE DATABASE Db1
SET DATABASE Db1
CREATE TABLE Tab1 (C1:REAL,C2:INT)
CREATE DATABASE Db2
SET DATABASE Db2
CREATE TABLE Tab1 (C7:CHAR(5),AA:VARCHAR(2))
CREATE TABLE Tab2 (Toto:CHAR(120))
LIST TABLES
DROP TABLE Tab1
LIST TABLES
SET DATABASE Db1
LIST TABLES
LIST DATABASES
DROP DATABASES
LIST DATABASES
```